Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

WordPress High-Availability: AWS Quick Start template

Autor: Andrés Bono Jiménez
Tutor: Francisco Javier Muñoz Calle

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

WordPress High-Availability: AWS Quick Start template

Autor:
Andrés Bono Jiménez

Tutor:
Francisco Javier Muñoz Calle

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Grado: WordPress High-Availability: AWS Quick Start template

Autor: Andrés Bono Jiménez
Tutor: Francisco Javier Muñoz Calle

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia
A mis maestros

Agradecimientos

Todo esfuerzo tiene su recompensa. El camino hacia esta, por supuesto, ha implicado momentos duros y sacrificados que no hubiesen podido superarse sin la ayuda y el apoyo de todos los que me han rodeado a lo largo de este viaje.

Por ello y, en primer lugar, quiero agradecer a mi familia y compañeros de clase, quienes me han apoyado en este largo proceso y me han animado en los momentos difíciles.

En segundo lugar, agradecer a todos los profesores que me han transmitido su conocimiento y experiencia a lo largo de estos años. En particular, a mi tutor Fco. Javier Muñoz, que me ha guiado y ayudado a culminar mi carrera con este proyecto.

Finalmente, quiero agradecer a Bitnami el haberme permitido llevar a cabo este proyecto con el que tanto he crecido profesional y personalmente.

Andrés Bono Jiménez
Estudiante del Grado en Ingeniería de las Tecnologías de Telecomunicación
Sevilla, 2019

Resumen

Solo un pequeño porcentaje de aplicaciones se han diseñado para funcionar de forma nativa con los servicios informáticos en la nube. Si se desea aprovechar el máximo potencial de plataformas que ofrecen infraestructura como servicio, es necesario realizar un proceso de migración que puede resultar complejo.

WordPress High Availability, dentro del marco de las soluciones *Quick Start* de *AWS*, se presenta como la solución de referencia propuesta por *Bitnami* junto con *AWS* y viene a solventar el problema mencionado de migración a la nube. Mediante el uso de *CloudFormation*, se describe una arquitectura de red que los usuarios pueden lanzar de forma sencilla utilizando los elementos de la nube que son necesarios para poder ofrecer la aplicación *WordPress* con características de alta disponibilidad.

Con el desarrollo de esta memoria se pretenden conseguir varios objetivos. Por una parte, se trata de introducir al lector en los conceptos principales sobre los elementos que componen la arquitectura de la solución. Por otra parte, se realiza una descripción de los pasos que se han de seguir para integrar una solución de estas características en los procesos de Bitnami. Esta integración incluye la construcción, el despliegue, el testeo y la publicación del producto, de forma que las tareas de mantener actualizados los componentes de WordPress High Availability pasen a ser un proceso automático.

Abstract

Only a small percentage of applications have been designed to work natively with cloud computing services. If platforms which offer infrastructures as a service want to be made the most of them, a migration process is necessary to be developed, which can come of very complex.

WordPress High Availability, within the framework of *Quick Start de AWS* solutions, is presented as the reference solution by *Bitnami* together with *AWS*, which solves the mentioned problem about migration to the cloud. Through the use of *CloudFormation,* we describe a network architecture which users can launch easily by using the necessary cloud elements in order to provide the *WordPress* application with high availability features.

The development of this project aims to achieve several objectives. On the one hand, the reader is introduced to the main concepts about the elements which form the architecture of the solution. On the other hand, it is presented a description of the steps which must be followed to integrate a solution of such characteristics in the *Bitnami's* processes. These include the stages of building, deployment, testing and publishing of the product so that the maintenance tasks needed to keep the *WordPress High Availability* components up to date become an automatic process.

Índice

Agradecimientos *ix*

Resumen *xi*

Abstract *xiii*

Índice *xv*

Índice de Tablas *xix*

Índice de Figuras *xxi*

1 Introducción *1*

1.1 Motivación, objetivos y plan de trabajo **2**

1.2 Situación actual **2**

1.3 Organización de la memoria **3**

1.4 Conceptos clave **3**
1.4.1 AWS 3
1.4.2 Bitnami 4
1.4.3 WordPress 4

2 Servicios informáticos en la nube *5*

2.1 Clasificación de plataformas de cloud **5**
2.1.1 Modelo según sus servicios 5
2.1.2 Modelo según su implementación 8

2.2 Características **8**

2.3 Cronología, origen e historia **11**
2.3.1 Cronología de AWS 13

2.4 Comparativa de plataformas Cloud **16**

2.5 Conclusiones **18**

3 Servicios de Amazon Web Services *21*

3.1 Regiones y zonas de disponibilidad **21**

3.2 Recursos **22**
3.2.1 Recursos EC2 22
3.2.2 Recursos RDS 25
3.2.3 Elastic Load Balancing 26
3.2.4 Escalado automático 28
3.2.5 Route 53 29
3.2.6 Gestor de certificados SSL 29

3.3 CloudFormation **31**
3.3.1 Características de CloudFormation 32
3.3.2 Estructura de las plantillas 33

3.3.3 Señalización 34
3.3.4 Alternativas globales: Terraform 34

3.4 Quick Starts 35
3.4.1 Qué son las soluciones Quick Start 35
3.4.2 Requisitos de las soluciones Quick Start 36

3.5 Otras herramientas de desarrollo 37
3.5.1 Herramienta por línea de comandos 37
3.5.2 Kits de desarrollo 38

4 Arquitectura de la aplicación 41

4.1 Migrar aplicaciones tradicionales a la cloud 41
4.1.1 Separación de los servicios en diferentes máquinas 42
4.1.2 Utilización de imágenes de instancias 43
4.1.3 Compartición de ficheros 45
4.1.4 Utilización de zonas de disponibilidad 48
4.1.5 Instancias bastión 50
4.1.6 Balanceo de carga 51

4.2 Diagrama de arquitectura 54

5 Funcionalidades principales y Validación de la solución 57

5.1 Despliegue sencillo de WordPress 58
5.1.1 Despliegue utilizando el portal de AWS 58
5.1.2 Despliegue utilizando aws-cli 59

5.2 Actualización de la solución 60

5.3 Copias de respaldo de la base de datos 63

5.4 Escalado horizontal de instancias. 63

5.5 Generación automática de certificados SSL 66

5.6 Alta disponibilidad 68
5.6.1 Validación 1: Eliminación manual de una instancia dentro del grupo de escalado automático. 68
5.6.2 Validación 2: Comparativa del tiempo medio de respuesta entre WordPress Stack y WordPress High Availability ante situaciones de alta demanda. 69

6 Componentes de la imagen (AMI) 73

6.1 Creación de la imagen 73

6.2 Configuración automática: Nami 74
6.2.1 Módulos de Nami 75

6.3 Provisionado automático: Provisioner 76
6.3.1 Introducción 76
6.3.2 Implementación de la cloud AWS en Provisioner 77
6.3.3 Definición de la información de la solución 80

6.4 Comunicación entre *CloudFormation* y las instancias 81
6.4.1 cfn-tools 81
6.4.2 *cloud-init* y *user-data* 83

6.5 Diagrama de componentes 85

7 Integración de la solución en los procesos de Bitnami 87

7.1 Construcción y despliegue 87
7.1.1 Modificación de la configuración de Jenkins 88
7.1.2 Módulo de BRadmin 91

7.2 Testeo 92
7.2.1 Tests funcionales 93

7.2.2 Tests de verificación 94
7.2.3 Tests de las plantillas 96

7.3 Publicación 96

8 Fases de desarrollo 99

8.1 Tecnologías y herramientas utilizadas 99

8.2 Entorno de trabajo 100
8.2.1 Organización 101
8.2.2 Software utilizado 101

8.3 Fases de desarrollo 103
8.3.1 Solicitud del proyecto y primeros pasos 103
8.3.2 Implementación de la cloud AWS en Provisioner 104
8.3.3 Construcción del provisioner-bundle 104
8.3.4 Primera versión de la solución 105
8.3.5 Reimplementación de la solución 107
8.3.6 Nuevas funcionalidades requeridas en Provisioner 109
8.3.7 Integración de la solución en los procesos de Bitnami 111
8.3.8 Estimación de costes 111
8.3.9 Desarrollo de la guía de despliegue 111
8.3.10 Ampliación de la batería de tests para WordPress 112
8.3.11 Lanzamiento y promoción de la solución 112

9 Conclusiones 115

Referencias 117

Anexo A: Código fuente implementado de Provisioner 121

Anexo B: Contenido de la definición de Provisioner 131

Anexo C: Primera versión de las plantillas de CloudFormation 135

Anexo D: Plantillas de CloudFormation 153

Anexo E: Configuración de Jenkins 213

Anexo F: Configuración de BRadmin 217

Anexo G: Tests Funcionales 223

Anexo H: Tests de verificación 229

ÍNDICE DE TABLAS

Tabla 2-1 Tipos de servicios de nube. 8
Tabla 2-2. Comparativa entre AWS, Azure, Google e IBM. Adopción y número de máquinas virtuales. Porcentaje base y crecimiento anual. [19] 18
Tabla 3-1 Recursos AWS utilizados. 30
Tabla 3-2. Representación de datos. JSON y YAML. 31
Tabla 3-3. Estructura básica de las plantillas CloudFormation. [37] 33
Tabla 3-4. Definición de instancia AWS mediante Terraform. 34
Tabla 3-5 Requisitos técnicos de las soluciones Quick Start 37
Tabla 4-1. Sección de la configuración de base de datos en WordPress 42
Tabla 4-2. Shell script que lanza una instancia EC2 y espera hasta su inicialización completa. 44
Tabla 4-3. Extracto de configuración del grupo de auto escalado para soporte de múltiples zonas de disponibilidad. 48
Tabla 4-4. Configuración del grupo de destino del balanceador de carga. 52
Tabla 4-5. Configuración del listener del balanceador de carga. 53
Tabla 5-4. Comparativa de características entre la stack de WordPress tradicional y la nueva solución WordPress High Availability. 57
Tabla 5-1. Estados de las instancias durante el proceso de actualización mediante Rolling Update. 62
Tabla 5-2. Parámetros de configuración de copias de respaldo del clúster RDS. 63
Tabla 5-3. Definición de política de escalado basada en uso de CPU. 63
Tabla 6-1. Contenido de un módulo de Nami. 75
Tabla 6-2. Estructura general del código de Provisioner. 76
Tabla 6-3. Bloque CloudFormation Init para paso de secretos. 82
Tabla 6-4. Extracto de receta de Provisioner para el envío de señal. 82
Tabla 6-5. Uso de user-data en la solución. 83
Tabla 7-1. Definición de parámetros de entrada de trabajos de Jenkins. 89
Tabla 7-2. Definición de un método que clona un repositorio de git. 90
Tabla 7-3. Modificación de las plantillas para inyección de AMI construida. 91
Tabla 7-4. Ejecución de test que añade un comentario a una entrada del blog. 93
Tabla 7-5. Función auxiliar para añadir comentario. 94
Tabla 7-6. Test de verificación para el chequeo de EFS. 95
Tabla 8-1. Fases de desarrollo del proyecto. 99
Tabla 8-2. Contenido del provisioner-bundle. 105
Tabla 8-3. Estructura final de las plantillas CloudFormation. 107

ÍNDICE DE FIGURAS

Figura 1-1. Fases del proyecto WordPress High Availability. 2
Figura 2-1. Tipos clásicos de servicios de nube. 6
Figura 2-2. Comparación de modelos de servicios de informática en la nube por capas que administrar. [4] 7
Figura 2-3. Consola de administración de AWS. 9
Figura 2-4. Consola de administración de Azure. 10
Figura 2-5. Consola de administración de GCP. 11
Figura 2-6. Error de pasarela del balanceador de carga de Azure obtenido en el navegador. 12
Figura 2-7. Error de pasarela del balanceador de carga de Azure utilizando curl. 12
Figura 2-8. Eje cronológico de la aparición de los proveedores cloud. 13
Figura 2-9. Cronología de servicios de AWS. 14
Figura 2-10. Número de zonas de disponibilidad por región (naranja) y próximas regiones (verde). [17] 15
Figura 2-11. Red global de ubicaciones de borde de Amazon CloudFront. [18] 16
Figura 2-12. Adopción de las cloud públicas en 2017 y 2018. [19] 17
Figura 2-13. Adopción de las cloud públicas en el ámbito empresarial durante 2017 y 2018. [19] 17
Figura 2-14. Porcentaje de encuestados agrupados por número de máquinas virtuales que su empresa gestiona. [19] 18
Figura 3-1. Jerarquía de nombrado de regiones. 22
Figura 3-2. Ejemplos de jerarquía de nombrado de regiones. N. California y Frankfurt. 22
Figura 3-3. Amazon EFS, MountTargets y configuración de red. 24
Figura 3-4. Múltiples direcciones IP para un mismo dominio de balanceador de carga. 26
Figura 3-5. Configuración de balanceo de carga. [32] 28
Figura 3-6. Interfaz de CloudFormation Designer. 32
Figura 3-7. Repositorio de plantillas Terraform para la cloud OCI. 35
Figura 3-8. Ejecución de comando describe-images con aws-cli. 38
Figura 3-9. SDKs de AWS para multitud de lenguajes de programación. [41] 39
Figura 4-1. Separación del servicio web y la base de datos. 42
Figura 4-2. Configuración con n servidores web accediendo a una misma base de datos. 43
Figura 4-3. Tiempo de ejecución de Shell script lanzando instancia EC2. 44
Figura 4-4. Descripción del sistema de ficheros y MountTargets con aws-cli. 46
Figura 4-5. Resolución DNS de los MountTargets en distintas zonas de disponibilidad. 47
Figura 4-6. Ejecución de comandos mount y df. 47
Figura 4-7. Instancias desplegadas en múltiples zonas de disponibilidad. 49
Figura 4-8. Diseño topológico de las subredes definidas por las plantillas de VPC. [43] 49
Figura 4-9. Diagrama de arquitectura de los equipos bastion. [44] 50
Figura 4-10. Vista de la consola de AWS mostrando instancias bastión. 51
Figura 4-11. Balanceador de carga como acceso al servicio. 51
Figura 4-12. Uso de la cookie AWSALB para el mantenimiento de conexiones con una misma instancia. 53
Figura 4-13. Diagrama de arquitectura de la solución WordPress High Availability. 55

Figura 5-1. Repositorio de código que contiene las plantillas de la solución WordPress High Availability. 58
Figura 5-2. Vista de parámetros a introducir para personalizar el despliegue. 59
Figura 5-3. Despliegue de la solución utilizando aws-cli. 60
Figura 5-4. Panel de administración de WordPress indicando que hay una nueva versión disponible. 61
Figura 5-5. Eventos de CloudFormation durante una actualización de instancias mediante Rolling Update. 62
Figura 5-6. Instalación y ejecución de la herramienta stress en las máquinas virtuales. 64
Figura 5-7. Estado inicial del grupo de escalado automático formado por 2 instancias. 64
Figura 5-8. Monitorización mediante CloudWatch del porcentaje de uso de CPU en las instancias del grupo de escalado automático. 65
Figura 5-9. Lanzamiento de una nueva instancia en el grupo de escalado automático a causa del aumento del uso de CPU. 66
Figura 5-10. Registros DNS de la solución. 66
Figura 5-11. Correo electrónico de confirmación para la validación de un certificado SSL. 67
Figura 5-12. Certificado SSL válido generado. 68
Figura 5-13. Una nueva instancia es lanzada en cuanto se manda la orden de terminación de la instancia. 69
Figura 5-14. Chrome Development Tools utilizado para analizar tiempos de carga. 70
Figura 5-15. Tiempos de carga en milisegundos frente a procesos de CPU en ejecución para las soluciones WordPress Stack y WordPress High Availability. 71
Figura 6-1. Ciclo de vida de un modulo de Nami. 75
Figura 6-2. *Ejecución del método de Provisioner _getUserDataScript.* 77
Figura 6-3. Ejecución del método de Provisioner getUserData. 78
Figura 6-4. Información recolectada por el agente de estadísticas. 79
Figura 6-5. Ejecución de los métodos de Provisioner _getInstanceIdentityDocument y getMetaData. 80
Figura 6-6. Viaje de parámetros desde el usuario a los servicios. 81
Figura 6-7. Ejecución de cfn-init. 82
Figura 6-8. Recepción de la señal SUCCESS en el panel de control de CloudFormation. 83
Figura 6-9. Acceso a user-data mediante API de AWS. 84
Figura 6-10. Diagrama de bloques que componen la AMI. 85
Figura 7-1. Generación de plantillas mediante BRadmin y ERB. 88
Figura 7-2. Vista de un trabajo de Jenkins ejecutado diariamente para la construcción y testeo de la solución Wordpress High Availability. 89
Figura 7-3. Parámetros de entrada en el trabajo de Jenkins. 90
Figura 7-4. Fases por las que pasa un trabajo de Jenkins. 91
Figura 7-5. Utilización de la consola de BRadmin para la ejecución del método load_balancers. 92
Figura 7-6. Resultado mostrado tras la ejecución de tests funcionales. 93
Figura 7-7. Salida tras la ejecución de los tests de verificación para el balanceador de carga y el sistema de ficheros compartido. 96
Figura 7-8. Proceso de publicación de AMIs y plantillas. 97
Figura 8-1. Vista en Phabricator de la User Story para WordPress High Availability. 104
Figura 8-2. Diagrama de arquitectura de la primera versión de la solución. 106
Figura 8-3. Reporte de la herramienta TaskCat. 108
Figura 8-4. Diagramas temporales de inicialización de la aplicación. 110
Figura 8-5. Contenido de user-data almacenado por cloud-init. 111
Figura 8-6. Promoción del lanzamiento de la solución en Twitter por parte de AWS. 113

1 INTRODUCCIÓN

Los tiempos en los que un sitio web era servido por un nodo único dentro de internet han quedado atrás. Cada vez es más extraño encontrar sitios temporalmente inactivos o bajo operaciones de mantenimiento. Hay sitios de gran envergadura que soportan picos de alta demanda sin dejar de ofrecer servicio y con unos retardos de carga que no parecen empeorar ante estas situaciones.

Se podría pensar que esta excelente calidad de servicio se consigue adquiriendo costosos centros de datos, o con análisis de tendencias de uso para predecir los mencionados picos de demanda y actuar en consecuencia. La realidad es que la *cloud* o nube ofrece unos servicios y herramientas que son claves para escalar aplicaciones manteniendo unos costes muy ajustados y reducidos.

La gran mayoría de empresas se han dado cuenta de ello y han comenzado a sustituir su propia infraestructura por el uso de plataformas cuyo servicio es el de ofrecer infraestructura hardware por un tiempo determinado. Estas plataformas se denominan *IaaS* (Infrastructure as a Service) y los ejemplos más representativos actualmente son *Amazon Web Services* (*AWS*), *Google Cloud Platform* (*GCP*) o *Microsoft Azure* (*Azure*).

Este estudio pretende desglosar por medio de qué técnicas, herramientas y arquitecturas se puede conseguir alta disponibilidad y buena calidad de servicio con el uso de la plataforma *AWS* (Amazon Web Services), utilizando como caso práctico la conocida aplicación de gestión de contenidos *WordPress*; una aplicación tradicionalmente concebida para ejecutarse en una única máquina (es decir, no es una aplicación distribuida) y que el personal de administración de sistemas de una determinada empresa tendría que instalar y configurar.

Gracias a la multitud de ventajas que ofrece la nube, se puede conseguir que el despliegue de un sitio web deje de ser un quebradero de cabeza y sea tan simple como rellenar un formulario y pulsar un botón. El mantenimiento del sitio también pasa a ser una tarea muy simple: El software siempre actualizado en su última versión, y con garantía de disponibilidad incluso ante situaciones de tráfico imprevisto o durante la actualización del sistema.

1.1 Motivación, objetivos y plan de trabajo

Como desarrollador de *Bitnami*, se quiere contribuir al ecosistema de *AWS* con una nueva solución de *WordPress* orientada a entornos de producción. Esta nueva solución se proveerá en forma de plantilla *CloudFormation* (ver sección 3.3) incluyendo los elementos de *cloud* necesarios para asegurar alta disponibilidad del servicio web.

En concreto, *WordPress High Availability* es la solución a desarrollar por solicitud del equipo *AWS Quick Start* (ver sección *3.4*) *a Bitnami*.

Respecto a los objetivos de este proyecto, el principal es el propio de desarrollo de la solución solicitada, de forma que esta termine siendo publicada por *AWS*. Para ello, la solución debe contar con las mejores características que se puedan añadir al producto, ya sean características referidas a la seguridad, al rendimiento o a la disponibilidad del servicio.

Puesto que también se va a requerir del mantenimiento de la solución por parte de *Bitnami* una vez que esta sea publicada, la solución debe integrarse en los procesos de *Bitnami.* Mediante automatización de cualquier tarea manual, se pretende ganar en eficiencia.

La *Figura 1-1* recoge las diferentes fases implicadas en el proyecto y que permiten conseguir los objetivos previamente mencionados.

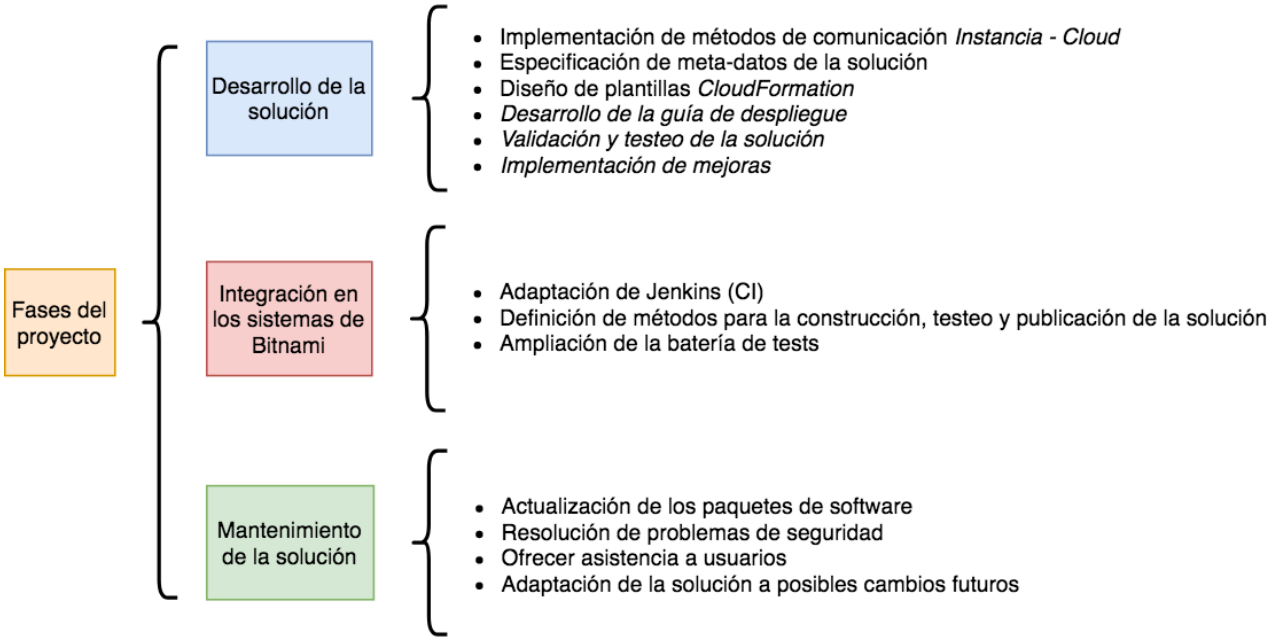


Figura 1-1. Fases del proyecto WordPress High Availability.

En la sección 8.3 se realiza una descripción pormenorizada de cada una de estas fases.

1.2 Situación actual

En el momento de redacción de este estudio, la situación de las tres mayores *clouds* con respecto a la disponibilidad de soluciones de *WordPress* configuradas para alta disponibilidad es bien distinta:

Por una parte, *AWS* no dispone de este tipo de solución. Proveerla es el objeto de este trabajo. Sí es cierto que hay algunas plataformas de terceros que usan AWS como infraestructura para ofrecer servicio. Este es el caso de https://pagely.com/wp-aws/.

Por otra parte, *Google Cloud Platform* cuenta con una solución llamada *Wordpress High Availability Beta*. Al estar todavía en desarrollo, carece de documentación adecuada sobre su despliegue y requiere de configuración adicional por parte del usuario tras ser lanzada.

Por último, *Azure* ofrece una solución (https://github.com/Azure/azure-quickstart-templates/tree/master/wordpress-mysql-replication) en la que la base de datos está replicada, pero que por su arquitectura, está lejos de poderse considerar una solución con garantías de alta disponibilidad.

1.3 Organización de la memoria

Este documento pretende dar a conocer el trabajo realizado para el desarrollo de la solución *WordPress,* dar a conocer las tecnologías *cloud* que se pueden aplicar en aplicaciones tradicionales y como se integra en los procesos de mantenimiento actuales de *Bitnami* una nueva solución tan diferente a las que normalmente se desarrollan.

Para ello, tras este capítulo, se introducen los conceptos principales de los servicios de *cloud* que existen, así como un recorrido histórico y una comparativa entre diferentes plataformas. Posteriormente, la descripción de estos servicios se concreta en la plataforma de trabajo de este proyecto: *AWS*.

Se continúa con la descripción de la arquitectura de la solución justificando la necesidad de cada uno de los recursos *cloud* utilizados. Para describir completamente la solución, también se detallan los componentes software que se incluyen en *WordPress High Availability*.

En un capítulo posterior se describe como se integra la solución en los procesos actuales de *Bitnami*. En este capítulo se incluyen detalles sobre la construcción y despliegue, testeo y publicación del proyecto.

Por último, se detallan las fases de desarrollo por las que pasa el proyecto, de forma que quede claro el alcance del trabajo realizado.

1.4 Conceptos clave

1.4.1 AWS

Amazon Web Services (*AWS*) es una organización que forma parte de *Amazon.com* y es actualmente la principal plataforma *cloud*. Ofrece infraestructura de red, capacidad de computación y servicios informáticos bajo demanda, de forma flexible y con tarificación según el uso de los recursos.

1.4.2 Bitnami

Bitnami es una compañía de desarrollo de software con sedes en Sevilla y San Francisco, así como empleados remotos por todo el mundo. Su misión es la de hacer que el software de servidores esté disponible para cualquiera en cualquier parte.

Como líder de la industria en cuanto a empaquetado de aplicaciones, su relación con *AWS* consiste en ofrecer un catálogo de aplicaciones (más de 150) que se publican en el *AWS Marketplace* (https://aws.amazon.com/marketplace) de forma que los usuarios de la *cloud* puedan utilizar estas aplicaciones de forma sencilla en forma de imágenes de *cloud*.

1.4.3 WordPress

WordPress es un sistema de gestión de contenidos (*CMS*) para la creación de cualquier tipo de página web. Está desarrollado en lenguaje *PHP* junto con la utilización de *MySQL* como base de datos. Se trata de la herramienta *CMS* más popular actualmente. Es un proyecto de software libre bajo la licencia *General Public License*. [1]

Respecto a las tecnologías mencionadas, PHP (PHP: Hypertext Preprocessor) es un lenguaje de programación de software libre ampliamente utilizado en desarrollo web que permite realizar un procesado en el lado servidor para generación de contenido dinámico.

MySQL por su parte es la base de datos utilizada por *WordPress*. Es un sistema de gestión de bases de datos de tipo *SQL* desarrollado por *Oracle*.

2 SERVICIOS INFORMÁTICOS EN LA NUBE

AWS (*Amazon Web Services*) define los servicios informáticos en la nube o *cloud computing* de la siguiente forma:

> *"La informática en la nube es la entrega bajo demanda de potencia informática, almacenamiento en bases de datos, aplicaciones y otros recursos de TI a través de Internet con un sistema de precios basado en el consumo realizado".* [2]

En la definición previa se mencionan dos conceptos clave: la "entrega bajo demanda" y "sistema de precios basado en el consumo realizado". Y es que la elasticidad y el pago por uso son dos de las características más interesantes de los proveedores de servicios *cloud*. De entre estos proveedores, *AWS* es el mayor de ellos.

2.1 Clasificación de plataformas de cloud

2.1.1 Modelo según sus servicios

Según el tipo de servicio que se ofrece, las plataformas *cloud* se pueden clasificar en cinco grupos [3]:

- **IaaS (Infrastructure-as-a-Service)**: Infraestructura como servicio. Se ofrece directamente capacidad de computación, uno o más servidores y almacenamiento. Permite olvidar la instalación y mantenimiento de los servidores físicos, pero a la vez se tiene un control completo sobre el software instalado en ellos.

  El ejemplo clásico de este servicio sería A*WS EC2* (*Elastic Compute Cloud*), permite crear máquinas virtuales para su posterior uso. Otras plataformas como *Microsoft Azure VM* (*Virtual Machines*) ofrecen el mismo servicio: https://azure.microsoft.com/services/virtual-machines/.

- **PaaS (Platform-as-a-Service)**: Plataforma como servicio. Ofrecen una interfaz en la que desarrollar y desplegar aplicaciones software. No se tiene acceso al servidor en el que se va a ejecutar dicho programa, sino que las aplicaciones se ejecutan en entornos que

disponen de los lenguajes de programación más comunes (*Python*, *Java*, *.NET*…). Los servicios *PaaS* conforman la porción más pequeña de entre todos los tipos de *cloud*.

Heroku (https://www.heroku.com) o *Google Apps Engine* (https://cloud.google.com/appengine/) son ejemplos de este tipo de servicio. Ofrece alojamiento de aplicaciones escritas en diferentes lenguajes y su ejecución para hacerla disponible a los usuarios a través de internet.

- **SaaS (Software-as-a-Service)**: Software como servicio. Se trata de productos software completos ya desarrollados de los que los clientes o usuarios finales pueden hacer uso. Conforman la mayor porción de la *cloud*.

  En esta categoría se pueden englobar servicios como *GSuite* de *Google* (https://gsuite.google.com) que ofrece servicio de correo electrónico, calendario, edición de documentos de texto y un largo etcétera.

Los tres grupos anteriores son los servicios clásicos de *cloud* (los que el *NIST*[1] reconoce) y uno podría sustentarse en el anterior a modo de pila (*Figura 2-1*).

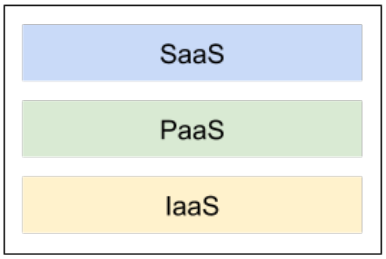


Figura 2-1. Tipos clásicos de servicios de nube.

Un diagrama que clarifica bastante los límites de cada modelo es el que se muestra en la *Figura 2-2*. En él se pueden apreciar los elementos que son necesarios administrar en servicios de nube virtualizados. A modo de modelo de capas, se muestra qué elementos deben ser administrados por el usuario y cuales por el proveedor de nube para cada tipo de modelo. Cuanto más a la derecha en el diagrama, menores son las tareas de mantenimiento, pero también es menor la personalización y el control sobre cada una de las capas.

[1] Instituto Nacional de Patrones y Tecnología de Estados Unidos.

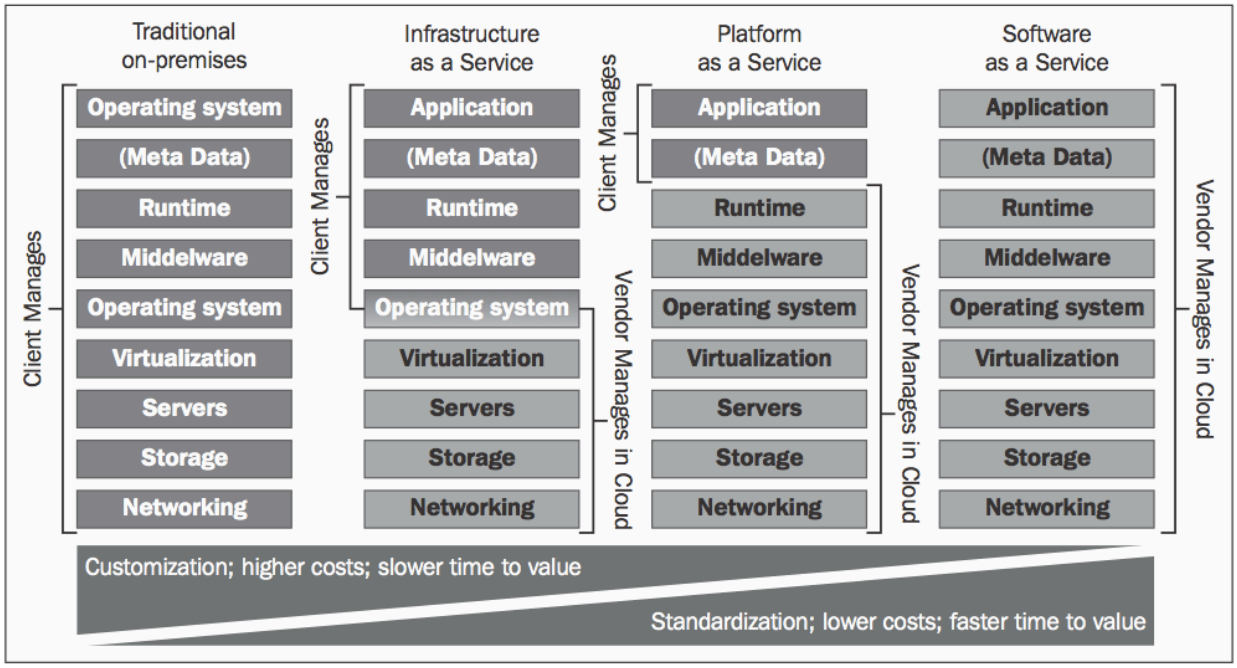


Figura 2-2. Comparación de modelos de servicios de informática en la nube por capas que administrar. [4]

Sin embargo, los tres modelos no son los únicos y cabe mencionar los siguientes:

- **CaaS (Container-as-a-Service)**: Contenedor como servicio. Se trata de una plataforma en la que poder gestionar y ejecutar contenedores[2] [5]. Estas plataformas se suelen integrar con otros servicios de *cloud* como son balanceadores de carga o volúmenes de almacenamiento compartidos que complementan las aplicaciones construidas con contenedores.

  La mayoría de *clouds* tienen servicios de este tipo ya que los contenedores son una tecnología en auge y en continuo crecimiento. Por mencionar algunos: *Amazon Elastic Container Service* (https://aws.amazon.com/ecs/) o *Google Kubernetes Engine* (https://cloud.google.com/kubernetes-engine/) basado en *Kubernetes*[3].

- **FaaS (Function-as-a-Service)**: Función como servicio. Es el grupo más reciente de servicio *cloud* y se requiere más recorrido para ver su evolución. Se ofrece la ejecución de un programa con una corta vida, ya que este tipo de servicios permiten ejecutar dicho programa (o función) una vez por petición de usuario. Esta es la gran diferencia con servicios de tipo *PaaS*, donde el programa debe ser diseñado para ser iniciado una sola vez y mantener su ejecución, solventando las posibles situaciones de concurrencia que se puedan dar. Este paradigma de una ejecución por petición permite una gran escalabilidad.

  Muchos casos que hacen uso de estos servicios están orientados al *IoT* (*Internet of Things*) ya que en general, los dispositivos son muy simples y con poca capacidad de computación por lo que solicitan procesamiento externo para realizar ciertas tareas. Por

[2] Un contenedor es un paquete autocontenido que agrupa ficheros como binarios ejecutables y librerías que se ejecuta de forma independiente y aislada del sistema operativo de la máquina host. Se podría decir que es un concepto parecido al de la virtualización, pero en el caso de los containers, el Kernel del sistema operativo es compartido con la máquina host y con otros contenedores que se puedan estar ejecutando. Las características que hacen interesantes a los contenedores son lo ligeros que son y su portabilidad por ser autocontenidos.

[3] Kubernetes es un proyecto de código libre para la orquestación de contenedores. Bitnami ha contribuido a este proyecto con funcionalidades como la de "sealed-secrets" (Consultar https://engineering.bitnami.com/articles/sealed-secrets.html para más información).

ejemplo, una cámara *IP* puede enviar una imagen capturada a un servicio *FaaS* para que realice alguna tarea de reconocimiento facial siguiendo un determinado algoritmo. [6]

Tabla 2-1 Tipos de servicios de nube.

Abstracción	Tipo de plataforma	Servicio que ofrece	Ejemplos
Menor	IaaS	Infraestructura	AWS EC2, Azure VM
	CaaS	Contenedor	AWS ECS, Google GKE
	PaaS	Plataforma	Heroku, Google Apps Engine
	FaaS	Función	AWS Lambda, Azure Functions
Mayor	SaaS	Software	Google GSuite, Dropbox

En la *Tabla 2-1*, a modo de resumen, se recoge la clasificación de servicios de nube, ordenándose de menor abstracción (más contacto y más control sobre los recursos desplegados) a mayor abstracción (requiere menos mantenimiento y configuración inicial).

2.1.2 Modelo según su implementación

Las *clouds* también se pueden clasificar en función de su implementación y forma de acceso. Se pueden diferenciar tres tipos [7]:

- **Pública:** Las aplicaciones y servicios residen en servidores de los que el usuario no es propietario, pero hace uso de ellos.
- **Privada:** También definida como "*on-premise"*, son centros de computación que el usuario administra y mantiene. A priori no son muy flexibles ya que para obtener más recursos no hay otra solución que realizar una inversión para aumentar la capacidad y número de los servidores.
- **Híbrida:** Se trata de una combinación de las anteriores. Parte de las aplicaciones utilizan recursos de nube pública y otra parte reside en equipos privados.

2.2 Características

Como se ha mencionado en la introducción de este capítulo, la elasticidad y el coste por tiempo consumido son dos de las características más llamativas del uso de servicios de *cloud*, pero no son las únicas. Vale la pena mencionar otras [4]:

- **Servicios bajo demanda:** Los clientes de proveedores de servicios de nube pueden crear los recursos que deseen sin necesidad de grandes conocimientos técnicos sobre como esos recursos están implementados. En general, la solicitud de esos recursos se hace por medio del panel de administración de una web que se suele denominar consola de administración.

  A modo de ejemplo, en *Figura 2-3*, *Figura 2-4* y *Figura 2-5* se pueden observar los diferentes paneles administración en *AWS*, *Azure* y *GCP* para la creación de un balanceador de carga *TCP*.

- **Accesibilidad a través de internet:** Los recursos son administrados y accesibles públicamente a través de internet soportándose una amplia variedad de clientes. Sin embargo, la implementación física y real de estos no es accesible por parte de los usuarios.

- **Recursos ilimitados:** Un usuario podría tener la percepción de que los recursos que se pueden crear son ilimitados siempre y cuando se pague por ellos. Evidentemente, la realidad no es esa, y los proveedores exprimen al máximo las capacidades de los equipos físicos, por medio de virtualización, sirviendo a múltiples usuarios con los mismos recursos físicos. A nivel lógico, se crean separaciones por seguridad.

- **Elasticidad, flexibilidad:** Los recursos se pueden crear y destruir bajo demanda e incluso de forma automatizada, basando estas decisiones en monitorización de tráfico o porcentaje de uso de *CPU*. De esta forma, de cara a los usuarios finales, se logra ofrecer el servicio deseado minimizando los costes en situaciones de baja demanda.

- **Registro y monitorización:** El uso de los recursos es medido y monitorizado de forma transparente para de esa forma poder cobrar por uso.

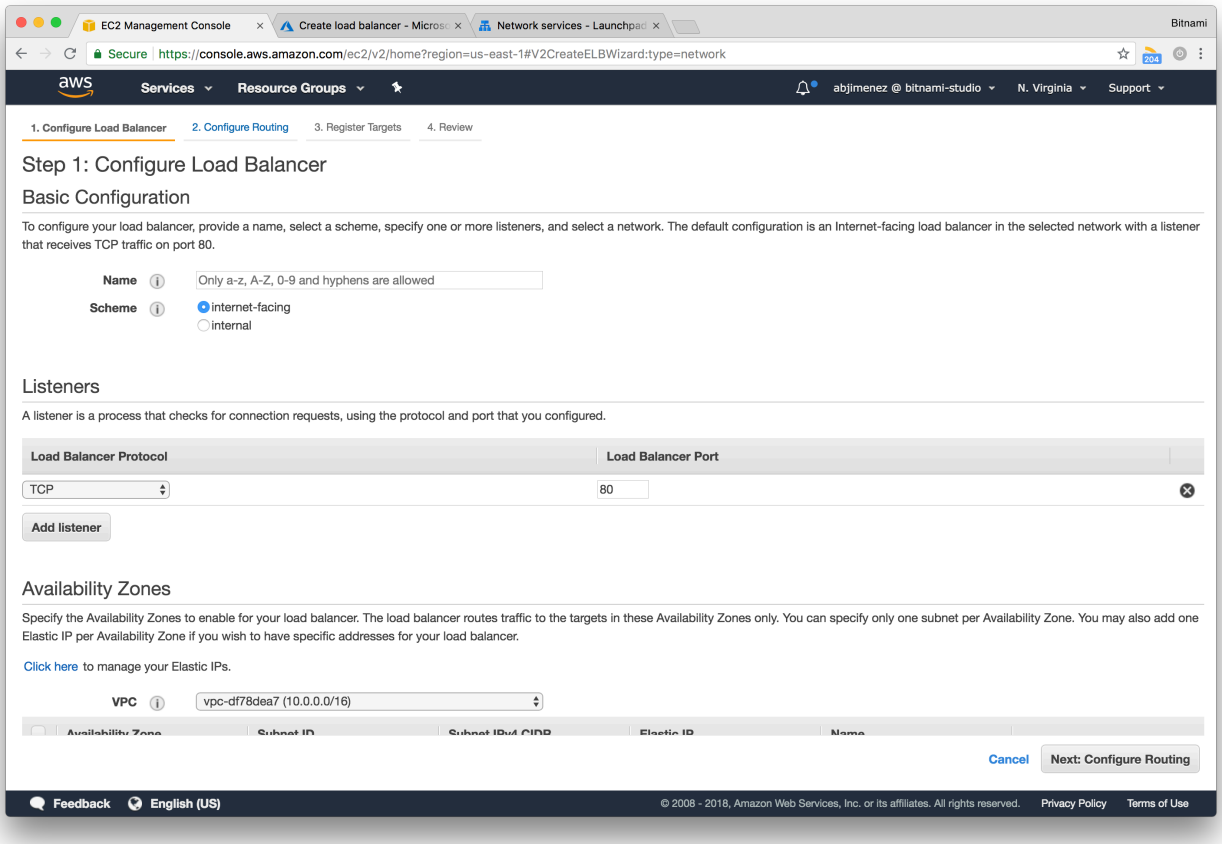


Figura 2-3. Consola de administración de AWS.

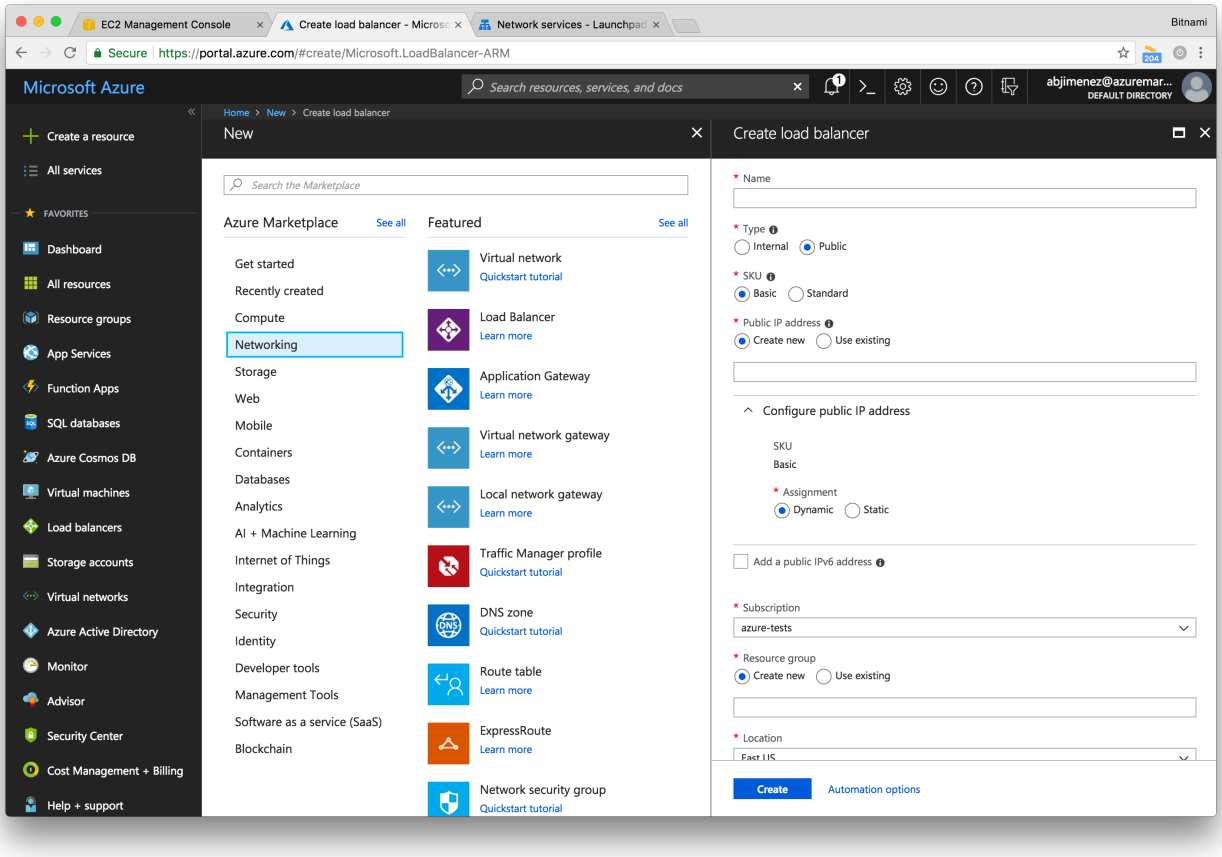


Figura 2-4. Consola de administración de Azure.

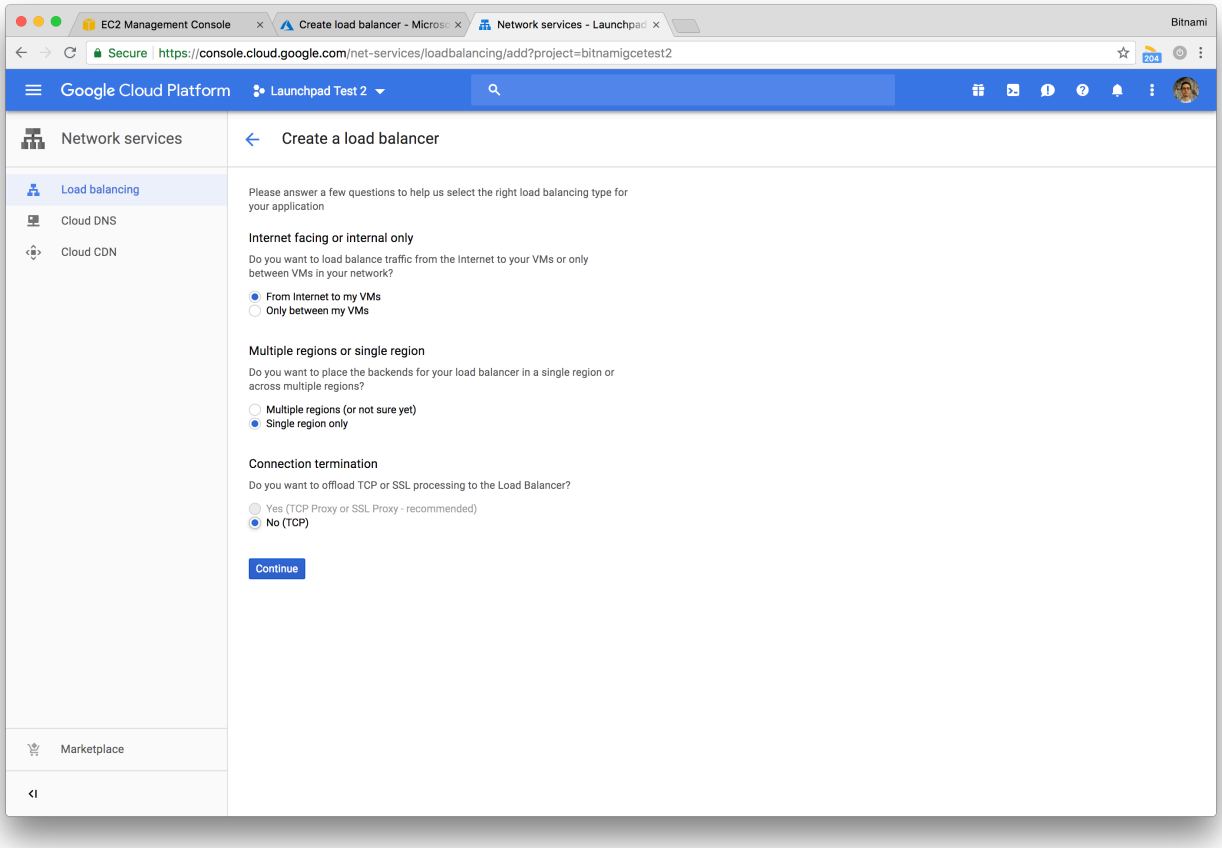


Figura 2-5. Consola de administración de GCP.

2.3 Cronología, origen e historia

En realidad, no ha habido una gran innovación en los servicios *cloud* que se ofrecen, puesto que la mayoría de ellos son tecnologías reconvertidas que ya se conocían. Sin ir más lejos, un ejemplo llamativo se da cuando la configuración de un balanceador de carga de la plataforma *Azure* es incorrecta y ningún equipo escucha las peticiones tras él. El mensaje de error *502* que se obtiene (*Figura 2-6*) resulta familiar. Utilizando una herramienta como *curl*[4] para obtener más información (*Figura 2-7*) se puede ver como el motor que hay detrás de este balanceador de carga no es más que el servidor web *IIS* (*Internet Information Services*) desarrollado por *Microsoft* que está redirigiendo las peticiones. Esto nos hace pensar que no estamos ante ningún elemento tecnológico que no conociésemos previamente.

[4] Curl es una herramienta de línea de comandos para la transferencia de datos. Soporta, entre otros, el protocolo HTTP.

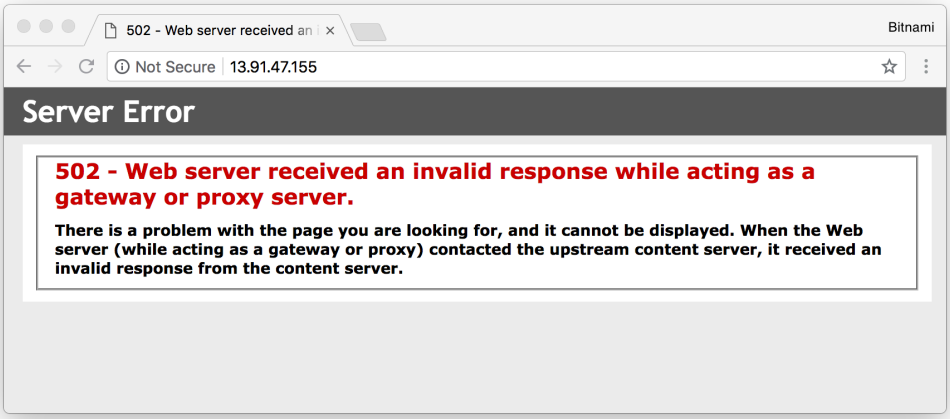


Figura 2-6. Error de pasarela del balanceador de carga de Azure obtenido en el navegador.

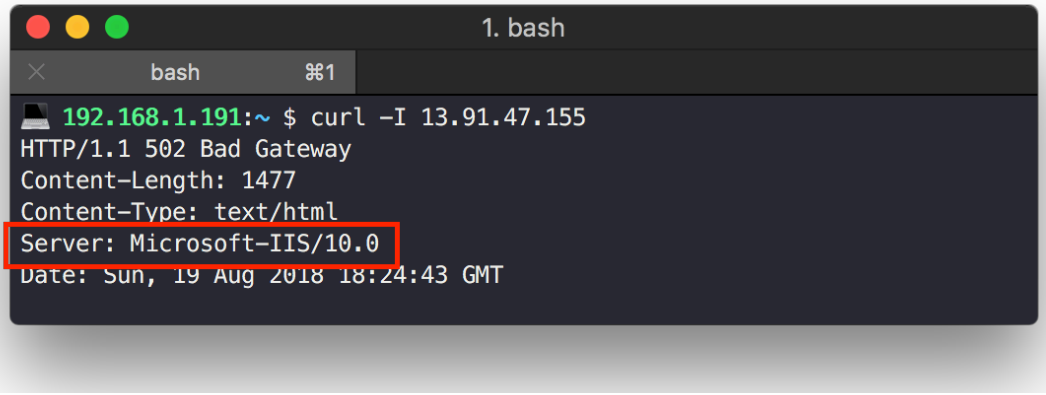


Figura 2-7. Error de pasarela del balanceador de carga de Azure utilizando curl.

Es difícil elegir un día en el que se empezó a hablar de la "nube" y como hoy en día se entiende puesto que no hubo una gran innovación tecnológica. La gran innovación se dio en la manera que esos servicios se hacen disponibles a los usuarios: de forma escalable, bajo demanda, y con pago por uso.

Ya en 1977, [8] el símbolo de una nube era utilizado para representar redes de ordenadores en esquemas elaborados por *ARPANET*[5]. Aunque hay documentos internos de 1996 [9] de la compañía *Compaq* que utilizaban "*cloud computing*", cuando el término empezó a popularizarse fue con la llegada en 2006 de los primeros servicios de *AWS* como *EC2* y el uso del término "*cloud computing*" por Eric Schmidt, antiguo director ejecutivo de *Google*, en una conferencia.

Salesforce[6], en 1999 [10], fue de las primeras empresas que empezó a seguir un modelo *cloud* en el que internet era el medio de distribución para ofrecer software a usuarios finales. Los programas se descargaban a través de internet bajo demanda.

En la *Figura 2-8* se recogen los diversos proveedores *cloud* por orden de aparición en un eje cronológico.

[5] Advanced Research Projects Agency NETwork. Fue la primera organización con una red de paquetes utilizando la familia de protocolos TCP/IP. Red predecesora de Internet.
[6] Empresa de software bajo demanda relacionado con el marketing y la relación con los clientes. Su producto más conocido es un CRM (Customer Relationship Management).

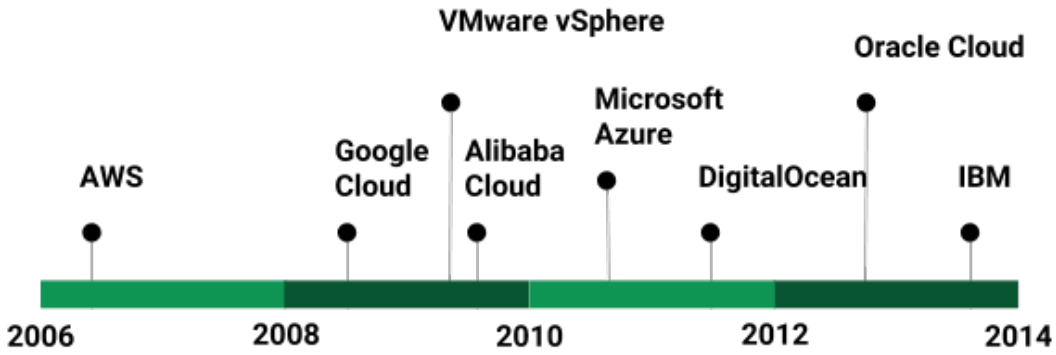


Figura 2-8. Eje cronológico de la aparición de los proveedores cloud.

2.3.1 Cronología de AWS

2.3.1.1 Servicios

Dado el crecimiento que la web de compras online *Amazon.com* estaba teniendo a finales de los 90 [11], se pensó en crear un equipo de ingenieros dentro de *Amazon* que se encargase de descomponer el sistema en partes más pequeñas e independientes que permitiesen a la web seguir escalando. Para ello, se empezó a trabajar en 2004 en *Amazon SQS* (*Amazon Simple Queue Service*), un servicio de paso de mensajes fiable que mantiene el orden.

El 14 de marzo de 2006 [11] se lanzó *Amazon S3* (*Amazon Simple Storage Service*), el primero de los servicios de computación de *AWS* que realmente estaba preparado para entornos de producción. En *Amazon* se preguntaron qué pasaría si se diese acceso a todo el mundo a la misma tecnología disponible para los ingenieros de *AWS*. Para algunos ese día cambió la forma de trabajar en las tecnologías de la información ya que se dio por primera vez acceso a un servicio de almacenamiento escalable y altamente fiable sin la necesidad de invertir grandes cantidades de dinero desde el primer día. El servicio era (y es) accesible desde internet por medio del protocolo *HTTP*.

A *Amazon S3* le siguieron otros servicios en los meses siguientes. *Amazon SQS* se lanzó públicamente en julio de 2006 [12] y *Amazon EC2* (*Amazon Elastic Compute Cloud*) en el siguiente año (versión beta). Aunque *S3* fue un servicio más revolucionario (nadie antes había resuelto el problema de almacenamiento "infinito"), *Amazon EC2* fue el servicio con más impacto. Se trata de un servicio en el que se paga por horas de computación. Basado en virtualización *XEN*[7] [13], se ofrecía un servidor *Linux* (o *Windows*) por un periodo de tiempo.

Estos tres servicios *S3*, *SQS* y *EC2*, formaron el núcleo de *AWS* como proveedor de servicios informáticos en la nube de tipo *IaaS*.

Posteriormente se pasó a publicar un sinfín de servicios complementarios. Por mencionar algunos de los más relevantes [14], [15], [16]:

- *Amazon SimpleDB* (2007), para servicios de base de datos.

[7] Es tipo de virtualización para sistemas x86 que permite la ejecución de varios sistemas operativos. Tiene buenos niveles de rendimiento y de aislamiento de recursos. Desarrollado por la universidad de Cambridge bajo la licencia GPL.

- *Amazon CloudFront* (2008), una red de distribución de contenidos.
- *Amazon Elastic Load Balancing* y *Amazon Auto Scaling* (2009) para el balanceo de tráfico y escalado del número de instancias.
- *Amazon Route 53* (2010), un servicio de *DNS*.
- *Amazon Simple Email* Service o *SES* (2011), servicio de envío de correo electrónico.
- *Amazon Redshift* (2012), un servicio de analítica de datos.
- *Amazon Kinesis* (2013), para análisis de datos en tiempo real.
- *Amazon Lambda (*2014), el servicio *FaaS*, funciones como servicio, de *AWS*.
- *Amazon IoT* (2015), una plataforma para la interacción de dispositivos relacionados con "Internet de las Cosas".
- *Amazon Elastic File System* o *EFS* (2016), Un sistema distribuido de almacenamiento de datos.
- *AWS Cloud9* (2017) un *IDE*[8] basado en la nube que permite escribir, ejecutar y depurar código.
- *Amazon Elastic Container Service* para *Kubernetes* o *EKS* (2018), proporciona un clúster de *Kubernetes* en *AWS*.

Figura 2-9. Cronología de servicios de AWS.

Es interesante ver cómo y en qué momentos *AWS* ha ido anunciando sus servicios (*Figura 2-9*). Se podría definir una primera etapa (aproximadamente 2004 – 2010) en la que el lanzamiento de servicios pretende dar cobertura a los servicios web más básicos: almacenamiento, bases de datos, redes de distribución de contenido, *DNS*…

Sin embargo, en la segunda etapa (aproximadamente 2011 - actualidad), por regla general, los servicios que se lanzan son más especializados en las tecnologías que en ese momento están en auge. Se pueden identificar servicios dedicados al "*Big Data*"[9] (2012 – 2013) o a "Internet de las

[8] Entorno de desarrollo integrado que facilita a los programadores la escritura de código.
[9] El Big data es un campo de la informática orientado al manejo de grandes cantidades de datos de cualquier tipo para su análisis y tratamiento de forma masiva.

Cosas" (2014 – 2015) que han sido dos de los campos de la tecnología con más impulso de los últimos años.

Actualmente en *AWS*, como se comunica en los diferentes *Summits*[10], las tendencias más recientes se dirigen hacia el desarrollo de servicios orientados a la inteligencia artificial, arquitecturas *serverless*[11] así como el uso tecnologías relacionadas con contenedores y *Kubernetes* como su reciente servicio *Amazon EKS*.

2.3.1.2 Expansión geográfica

Desde el inicio de *AWS*, no solo se han ido añadiendo más productos y servicios, también se ha hecho una expansión en las localizaciones geográficas de estos servicios. *AWS* se divide en diferentes regiones y actualmente soporta 19 [17], incluyendo regiones de tipo gubernamental (*Figura 2-10*). A esto hay que sumar los 132 puntos de presencia[12] de *CDN* [18], en 59 ciudades de 26 países (*Figura 2-11*).

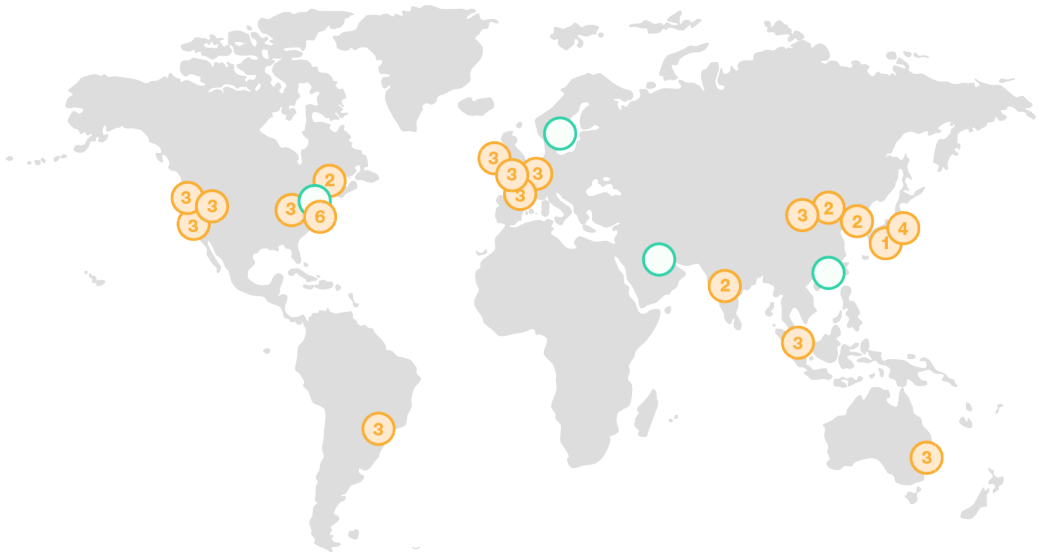


Figura 2-10. Número de zonas de disponibilidad por región (naranja) y próximas regiones (verde). [17]

[10] Los AWS Summits son conferencias que Amazon organiza por todo el mundo para dar a conocer las novedades y tendencias con respecto a sus servicios. Uno de ellos se celebra anualmente en Madrid, al que pude asistir.
[11] Las arquitecturas serverless hacen uso de servicios FaaS para ejecutar los programas. Son arquitecturas que permiten una gran escalabilidad.
[12] Elemento de las redes de distribución de contenido que replica la información de un servidor central y hace que el acceso a esta sea más eficiente para clientes cercanos.

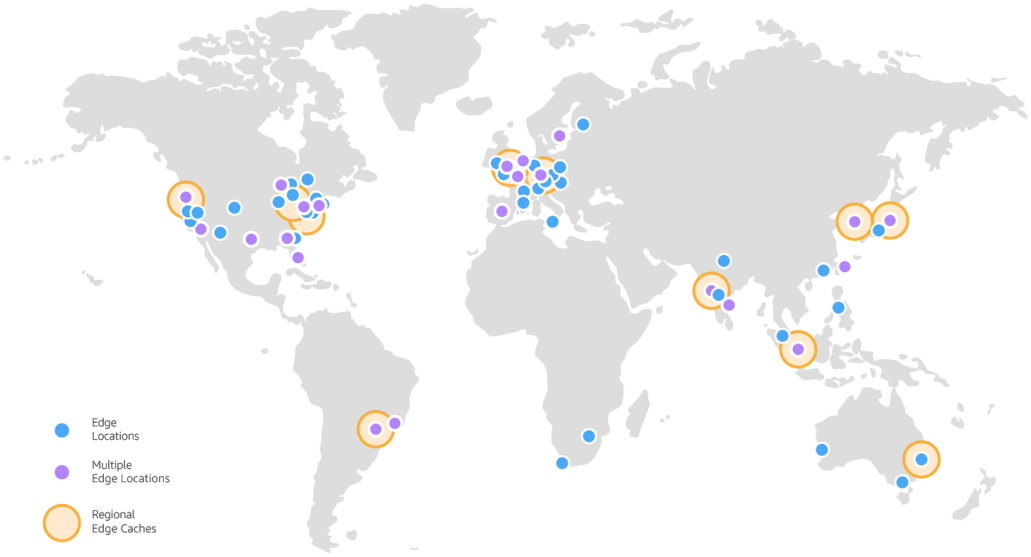


Figura 2-11. Red global de ubicaciones de borde de Amazon CloudFront. [18]

2.4 Comparativa de plataformas Cloud

Si se clasifican las diferentes plataformas cloud en grupos según su importancia, los tres mayores proveedores son *Amazon Web Services*, *Microsoft Azure* y *Google Cloud Platform*. Tras estos se encuentran otros como *Oracle Cloud Infrastructure*, *Alibaba Cloud*, *IBM Cloud* y *Vmware Cloud*. Por último, existen otros muchos con menor cuota de mercado. Por poner algunos ejemplos: *DigitalOcean*, *Cisco dCloud* o *Huawei Cloud*.

RightScale publica anualmente un reporte sobre el estado de la *cloud* basado en una encuesta a profesionales técnicos de diversas empresas del sector. Los datos mostrados a continuación se basan en el reporte de 2018. [19]

Algunas de las conclusiones más relevantes son que *AWS* sigue siendo líder en el mercado de las *clouds* (*Figura 2-12*), pero otros proveedores están creciendo más rápidamente. Es de especial importancia el caso de *Azure* en el entorno empresarial como se puede ver en la *Figura 2-13*.

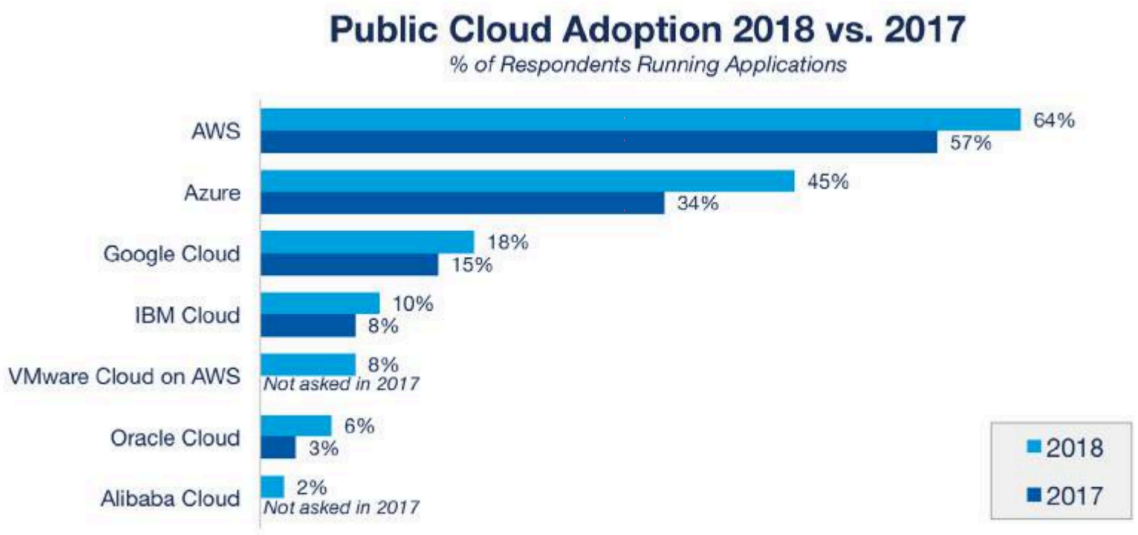


Figura 2-12. Adopción de las cloud públicas en 2017 y 2018. [19]

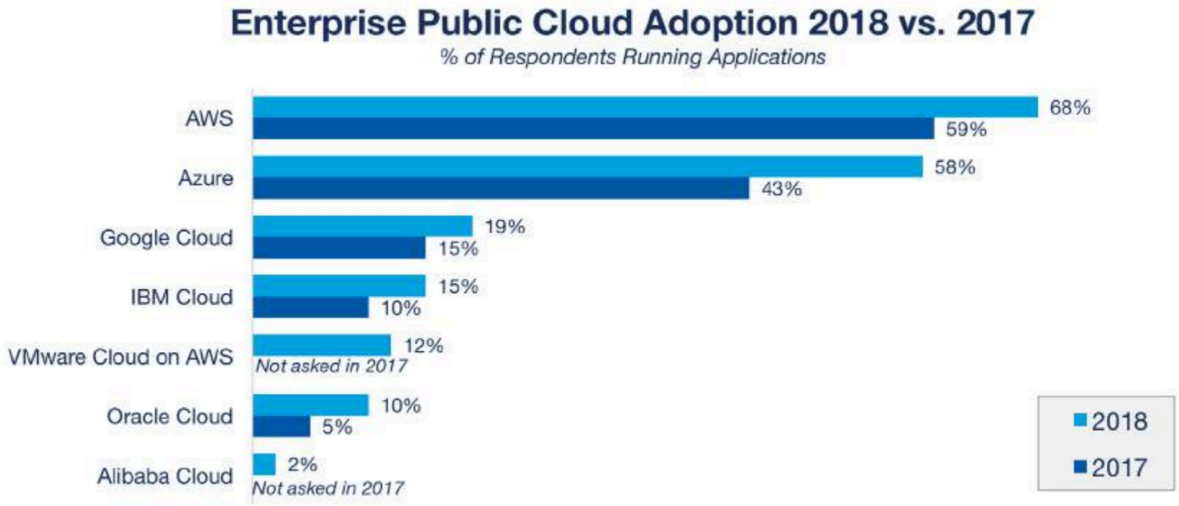


Figura 2-13. Adopción de las cloud públicas en el ámbito empresarial durante 2017 y 2018. [19]

Es importante destacar que es muy frecuente que los usuarios de *cloud* no usen un único proveedor, es por eso que la suma de los porcentajes no es el 100%. Es más, el 81% de las empresas de más de 1000 empleados hacen uso de más de una *cloud*.

Si se observa el número de máquinas virtuales por *cloud* (*Figura 2-14*), se aprecia un fenómeno interesante. *VMware vSphere*[13] es líder en esta métrica puesto que las empresas de un 15% de los encuestados tiene más de 1000 máquinas virtuales en esa plataforma. Esto es debido a que tradicionalmente en el entorno empresarial, el entorno que mayor número máquinas gestiona, no se ha querido hacer uso de las *clouds* públicas. Esta tendencia está cambiando.

[13] VMware vSphere es un proveedor de cloud privada orientado a virtualizar centros de datos.

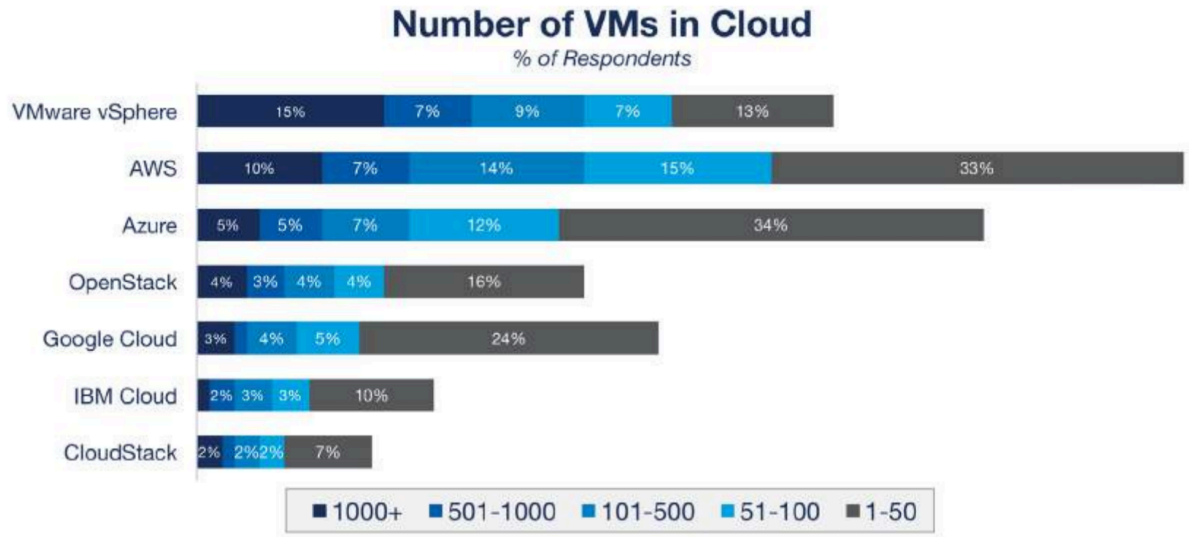


Figura 2-14. Porcentaje de encuestados agrupados por número de máquinas virtuales que su empresa gestiona. [19]

En la *Tabla 2-2* se recogen datos comparativos entre los mayores proveedores *cloud*. *AWS* sigue siendo líder, pero el resto crece más rápidamente.

Tabla 2-2. Comparativa entre AWS, Azure, Google e IBM. Adopción y número de máquinas virtuales. Porcentaje base y crecimiento anual. [19]

AWS vs. Azure vs. Google vs. IBM Enterprise Scorecard

Area	AWS	Azure	Google	IBM
% Adoption	68%	58%	19%	15%
YoY Growth in Adoption	15%	35%	26%	50%
% Adoption in Beginners	47%	49%	18%	14%
% with Footprint >50 VMs	58%	44%	17%	14%
YoY Growth in Footprint> 50 VMs	14%	38%	42%	56%

AWS leads
Other vendors lead AWS
Source: RightScale 2018 State of the Cloud Report

2.5 Conclusiones

Como se puede ver, los servicios informáticos en la nube son mucho más que máquinas virtuales. Se trata de una nueva forma de ofrecer recursos de computación como si fuesen servicios.

A la hora de desplegar aplicaciones, las inversiones iniciales se hacen menores puesto que el pago de los recursos se hace por tiempo. Escalar los sistemas es mucho más fácil y esa agilidad es una gran ventaja competitiva. Por otra parte, no tener que mantener el hardware elimina un gran número de quebraderos de cabeza para los profesionales de las tecnologías informáticas.

3 SERVICIOS DE AMAZON WEB SERVICES

De todas las *clouds* analizadas en el capítulo 2, la plataforma en la que se basa este proyecto es *AWS*. Para el desarrollo de la solución *WordPress High Availability* ha sido necesario el uso de múltiples herramientas, tecnologías y servicios de *Amazon*, los cuales se describen a continuación, organizados por recursos individuales y una introducción a los conceptos de *CloudFormation* y *Quick Starts*.

En capítulos posteriores se explica como se agrupan estos elementos para dar forma a la solución final *WordPress High Availability*.

3.1 Regiones y zonas de disponibilidad

AWS dispone de infraestructura en multitud de localizaciones geográficas por todo el mundo. La infraestructura está organizada por regiones y estas a su vez subdivididas en zonas de disponibilidad. La existencia de regiones permite que los usuarios elijan donde quieren situar sus servidores, ya sea por cuestiones administrativas y legales o para tener una menor latencia cuando se accede a estos.

Las zonas de disponibilidad se caracterizan por estar aisladas unas de otras [20], de forma que fallos en una de ellas no implican fallos en las demás. Esta característica es clave para poder ofrecer alta disponibilidad en un servicio. Otra característica que hay que destacar es que las zonas de disponibilidad dentro de una misma región se pueden comunicar de forma más eficiente en tiempo y también en dinero, ya que el tráfico de datos dentro de una misma región es gratuito [21].

Existen también regiones de tipo gubernamental [22] que están dedicadas, por el momento, a organismos públicos del gobierno de Estados Unidos. Esta separación en una región diferente se hace para alojar datos sensibles y asegurar las garantías de seguridad más exigentes.

Las regiones siguen una jerarquía de nombrado. Como nivel superior se define el continente o país al que pertenecen, seguido por un punto cardinal o subzona, un índice que hace referencia a alguna ciudad donde se encuentra el centro de datos y por último una letra haciendo referencia a la zona de disponibilidad. Las figuras *Figura 3-1* y *Figura 3-2* ilustran esta idea.

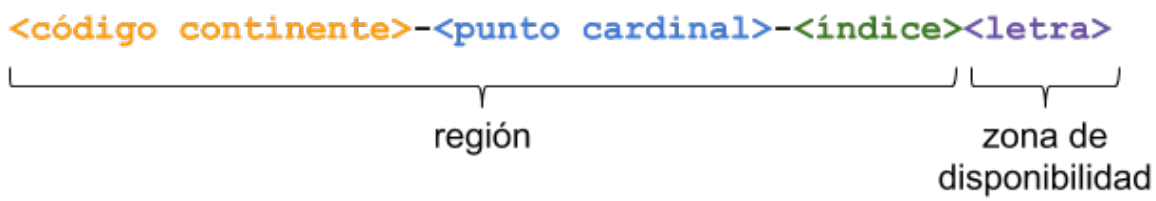


Figura 3-1. Jerarquía de nombrado de regiones.

Figura 3-2. Ejemplos de jerarquía de nombrado de regiones. N. California y Frankfurt.

3.2 Recursos

A continuación, se describen los recursos más importantes de la plataforma *AWS* y que han sido utilizados en el proyecto. Cada uno de estos recursos es explicado en detalle en la guía de usuario "*Amazon EC2: User Guide for Linux Instances*". [23]

Como resumen de los recursos descritos y utilizados, se puede consultar la *Tabla 3-1*.

3.2.1 Recursos EC2

En este apartado se describen los recursos pertenecientes al servicio *EC2* (*Elastic Compute Cloud*). Son los tipos de recursos más básicos.

3.2.1.1 AMI

Una *AMI* es una imagen de máquina de *Amazon* (*Amazon Machine Image*) que contiene todo el software que una máquina virtual va a necesitar cuando sea lanzada. A partir de una misma *AMI* se pueden lanzar múltiples instancias. Estas imágenes están disponibles en el catálogo de *Amazon* https://aws.amazon.com/marketplace que *AWS* y sus colaboradores mantienen actualizado.

3.2.1.2 Volumen EBS

Un volumen *EBS* (*Elastic Block Storage*) es un dispositivo de almacenamiento por bloques que se utiliza como disco duro de las máquinas virtuales. Su tamaño es flexible puesto que puede ser modificado después de haberlo creado. Existen varias categorías, de tipos magnéticos o *SSD*[14]. Debe existir al menos uno por máquina virtual (el disco raíz) pero puede haber más.

[14] SSD (Solid-State Drive) es una tecnología de almacenamiento de datos no volátil.

3.2.1.3 Instancia

También denominadas máquinas virtuales (*VMs*) son uno de los recursos básicos más utilizados. Forman parte de la familia de recursos de *EC2* (*Elastic Compute Cloud*).

Cuando se crea una instancia del sistema operativo elegido, se crea un servidor virtual con acceso remoto en el cual se pueden instalar nuevos programas y aplicaciones, utilizar la capacidad de computación que esté disponible (según el tipo de instancia elegida) y ejecutar servicios. Estas instancias vienen acompañadas de un volumen *EBS* raíz, donde está incluido el sistema operativo.

Este tipo de recurso es configurable mediante ciertos parámetros. Entre los más importantes se encuentran el identificador de imagen (*AMI*), el tipo y tamaño de instancia, el par de claves *SSH*[15] pública/privada para garantizar el acceso remoto o la *user-data* (concepto que se describe en detalle en capítulos posteriores).

3.2.1.4 IP elástica

Este tipo de recurso no es más que que una dirección *IPv4* estática y reservada que es única y accesible de forma pública. Estas *IPs* se pueden a asociar a varios tipos de recursos como instancias o balanceadores de carga. En caso de no asignar una *IP* elástica a una instancia, a esta se le asocia una *IP* que también es pública pero no está reservada, es decir, la dirección podría cambiar y podría reasociarse a otro recurso que no tiene porqué estar bajo control. Por este motivo, las *IPs* elásticas son las recomendables para ser utilizadas en registros *DNS*[16].

Estas *IPs* se denominan elásticas ya que una vez se reservan, se pueden asociar y reasociar de forma rápida. Esto es muy conveniente, por ejemplo, ante una situación de fallo. La *IP* puede reasociarse a un recurso que si funcione correctamente y de esa forma seguir ofreciendo servicio.

3.2.1.5 VPC

Una *VPC* (*Virtual Private Cloud*) es una partición de la cloud (en concreto a una única región) para uso particular y que está aislada de forma lógica del resto de la nube. Se le asigna un rango de direcciones *IP* privadas y se puede dividir en subredes siguiendo la jerarquía de particiones por bloques de *IPs*.

Cualquier recurso con comunicación de red debe pertenecer a una *VPC*.

3.2.1.6 Grupo de seguridad

Son reglas de seguridad que se utilizan como un cortafuegos virtual para controlar el tráfico entre recursos dentro de una *VPC*. Se pueden definir reglas por protocolo, 6 (*TCP*), 17 (*UDP*), 1 (*ICMP*), por puerto (capa de transporte) así como por origen o destino *IP* de los paquetes.

Este tipo de grupos de seguridad no es un sustituto del propio cortafuegos que el sistema operativo de una instancia implemente[17]. Esto quiere decir que para que un paquete consiga

[15] SSH (Secure SHell) es el protocolo utilizado para la conexión remota segura entre máquinas Linux.
[16] DNS (Domain Name System) es un sistema distribuido de resolución de nombres de dominio a direcciones IP.
[17] En sistemas Linux, el cortafuegos, *netfliter* o derivados más modernos, forma parte del kernel y se controla por medio de herramientas como *iptables* o *firewalld*.

conectar una determinada instancia debe pasar dos filtros: el del grupo de seguridad y el del cortafuegos que implemente la instancia.

3.2.1.7 Sistema de ficheros elástico

También llamado *EFS* (*Elastic File System*). Este tipo de recurso crea un sistema de archivos compartido que se puede montar en múltiples instancias con acceso de lectura y escritura concurrente a los ficheros, como si se tratase de ficheros locales. Este sistema es compatible con el protocolo *NFS*[18] (*Network File System*) versiones 4.0 y 4.1 (*NFSv4*). El tamaño del sistema de ficheros no tiene porqué ser definido en la creación del recurso ya que este recurso es elástico en ese sentido.

Para el funcionamiento de este sistema de ficheros [24] es necesario crear uno o más recursos auxiliares denominados *MountTargets* o destinos de montaje, que proporcionan una *IP* que se puede utilizar para montar el sistema de ficheros mediante el comando de *Linux mount*. Se debe crear un destino de montaje por cada una de las zonas de disponibilidad existentes, de forma que se garantice la alta disponibilidad (*Figura 3-3*).

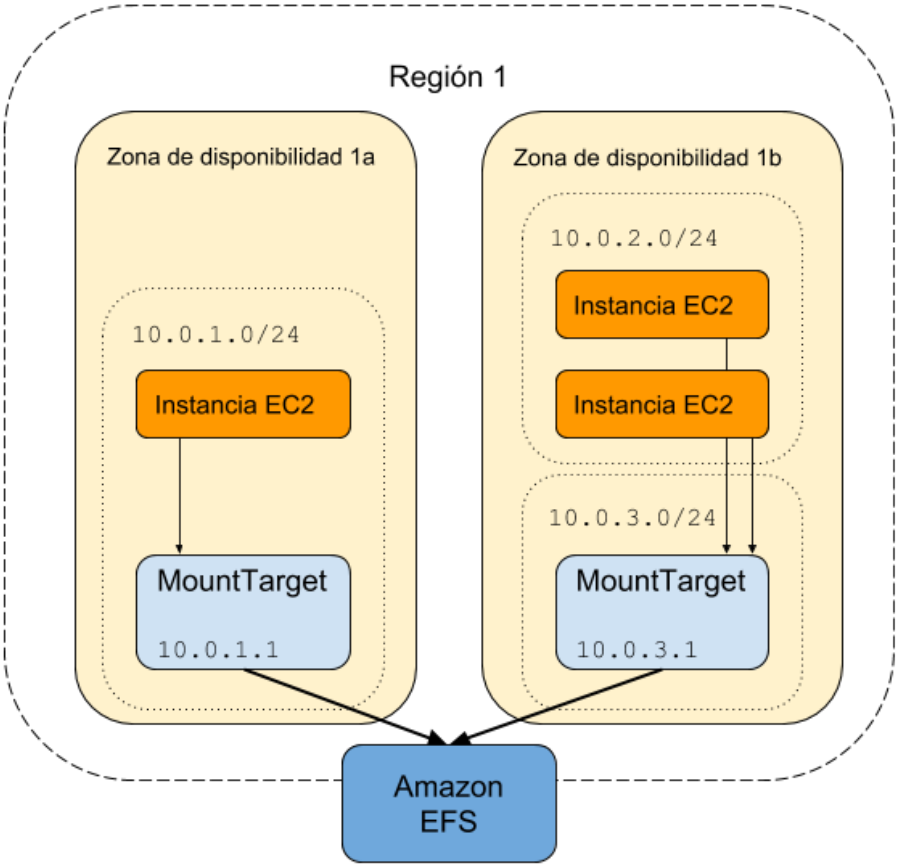


Figura 3-3. Amazon EFS, MountTargets y configuración de red.

Por otra parte, puesto que se trata de un recurso con el que se va a interactuar mediante *TCP*/*IP*, es necesario tenerlo en cuenta a la hora de configurar los grupos de seguridad (puerto 2049).

[18] NFS (Network File System) Es un protocolo utilizado en sistemas de archivos compartidos.

3.2.2 Recursos RDS

Los servicios de base de datos se ofrecen en *AWS* mediante un servicio distinto al de *EC2*. Para bases de datos de tipo relacional, este servicio es *RDS* (*Relational Database Service*).

Al ofrecerse el servicio de base de datos, el usuario se ahorra el mantenimiento de esta. Algunas de estas tareas son [25]:

- Copias de seguridad. Pueden ser programadas y periódicas o realizarse de forma manual.
- Aplicación de parches del sistema y del motor de bases de datos.
- Facilidad para escalar la base de datos, tanto en tamaño de almacenamiento como en capacidad de computación sin necesidad de realizar migraciones de datos.
- Configuraciones de alta disponibilidad teniendo una instancia principal del servicio en una zona de disponibilidad y otra secundaria en otra zona. En caso de producirse algún tipo de error, *RDS* cambia la instancia a la que se realizan las solicitudes de forma automática.
- Aumento del número de instancias de acceso para lectura para aumentar el rendimiento.

Amazon RDS ofrece los siguientes motores de bases de datos relacionales: *MySQL*, *MariaDB*, *PostgreSQL*, *Oracle*, *Microsoft SQL Server* y *Amazon Aurora*. Este ultimo motor es desarrollado por *Amazon* y ofrece una interfaz totalmente compatible con *MySQL* pero que se integra de una mejor forma con la infraestructura de *Amazon* obteniendo así un mayor rendimiento por un coste menor. Esto se traduce en cifras de más de 500.000 operaciones *SELECT* por segundo y más de 100.000 operaciones *UPDATE* por segundo[19] [26], [27], un rendimiento 5 veces mejor que el de *MySQL* ejecutándose en el mismo hardware.

3.2.2.1 Instancia de base de datos

Se trata del elemento básico de *RDS*. No es más que una máquina virtual o instancia preconfigurada para ejecutar un motor de bases de datos. Sus parámetros más importantes son el motor y versión de base de datos, ya que solo se puede ejecutar un tipo de base de datos por instancia, el tamaño (tipo) de servidor o el tipo de disco de la instancia.

3.2.2.2 Clúster de base de datos

Cuando se quieren agrupar múltiples instancias para formar un clúster de base de datos, se utiliza este tipo de recurso. En él se pueden definir propiedades comunes a todas las instancias como por ejemplo las copias de seguridad que se quieren realizar o las credenciales de acceso a la base de datos.

Otra característica interesante es que al definir un clúster y añadirle instancias a este, se consigue tener un único punto de acceso al clúster y las peticiones serán redirigidas a las instancias según convenga, para solventar situaciones de fallo.

[19] SELECT y UPDATE son las dos operaciones básicas en bases de datos relacionales para la lectura y escritura de información.

3.2.3 Elastic Load Balancing

Como su propio nombre indica, este grupo de recursos es el encargado del balanceo de carga de tráfico entrante. Esto quiere decir que las peticiones que se reciben en un punto de la infraestructura pueden ser redirigido atendiendo a ciertos tipos de reglas para poder así atenderlas de forma más rápida y eficiente.

Otra de las características importantes de los balanceadores de carga es que permiten escalar el número de servidores que atienden las peticiones sin cambiar el punto de acceso al que acuden los clientes.

Por último, se puede elegir que las peticiones vayan a servidores de varias zonas de disponibilidad. Esto implica una ventaja en cuanto a asegurar la disponibilidad del servicio.

Existen varios tipos de balanceadores de carga según el tipo de tráfico que se quiera redirigir. El tráfico balanceado puede ser de tipo *TCP* pensado para aplicaciones que implementan su propio protocolo de nivel de aplicación sobre el nivel de transporte. También existen balanceadores de tráfico *HTTP*[20] y *HTTPS* (nivel de aplicación). Estos son de alto nivel pudiendo administrar características del nivel de aplicación como son el manejo de sesiones o la terminación de conexiones *SSL*.

Respecto a los últimos, los balanceadores de carga de nivel de aplicación, existen tres componentes principales: El balanceador en sí, los grupos de destino y las reglas (*listeners*).

3.2.3.1 Aplication Load Balancer

Un balanceador de carga de nivel de aplicación redirige tráfico entrante a ciertos grupos de destino como pueden ser instancias *EC2* con un servicio web escuchando. Puesto que el tráfico de un determinado servicio puede ser variable, el balanceador de carga no está compuesto por un único nodo. Esta multiplicidad de nodos se resuelve mediante la traducción *DNS* del dominio asociado al balanceador, de forma que distintos clientes pueden obtener direcciones *IP* diferentes para un mismo dominio (*Figura 3-4*), es decir, para un mismo nombre de dominio, pueden existir múltiples *IPs* [28].

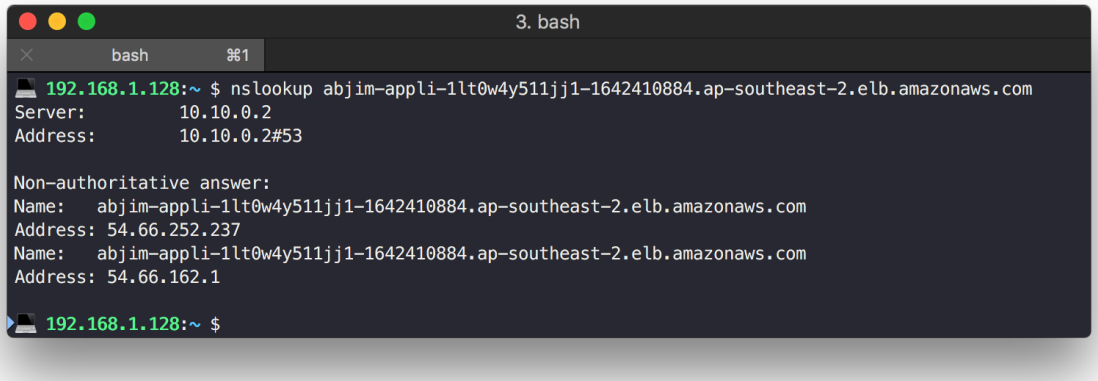


Figura 3-4. Múltiples direcciones IP para un mismo dominio de balanceador de carga.

[20] HTTP o HyperText Transfer Protocol es un protocolo de nivel de aplicación ampliamente utilizado en internet orientado al traspaso de contenido web. Existe una version segura que implementa cifrado basado en SSL/TLS (capa de transporte junto con TCP).

Tras haber resuelto la dirección, una vez que la solicitud del cliente alcanza uno de los nodos del balanceador, este debe decidir que qué instancia redirigirla. Para ello se utiliza el algoritmo *Round Robin*, que asigna de forma equitativa las peticiones. Otro punto importante que se tiene en cuenta es que una misma conexión *TCP* se mantiene con el mismo nodo de destino hasta que esta conexión termine, haciendo posible que se mantengan las sesiones.

Los parámetros más significativos del recurso *Load Balancer* son el esquema (interno o público si se desea exponer a internet), las subredes o zonas de disponibilidad y el tipo de balanceador (de aplicación o *TCP*) [29].

3.2.3.2 Grupo de destino

Para definir los destinos del tráfico a los que el balanceador de carga redirige, se utilizan los grupos de destino o *Target Groups*. Cada uno de ellos implementa un tipo de chequeo de estado (*health check*) y lo comunica al balanceador, de forma que, en caso de fallo, no se redirijan peticiones a nodos que no están disponibles. Para cada uno de los grupos de destino existe un *listener* con una determinada regla que hace que los paquetes de los clientes lleguen a ese grupo de destino.

Un concepto importante de los grupos de destino es el de las *Sticky Sessions*. Se trata de un mecanismo mediante el cual solicitudes pertenecientes a una misma sesión *HTTP*, son servidas por el mismo destino. Esta característica es configurable y se basa en la generación de una *cookie*[21] que envía al cliente [30].

3.2.3.3 Listener

Los *listeners* son entidades que esperan a los mensajes que el balanceador de carga recibe por parte de los clientes y basándose en una determinada regla definida, se redirigen a un grupo de destino. Cada *listener* puede tener una o más reglas. Cada una de estas reglas definen un grupo de destino, una condición y también una prioridad [31]. En el caso de los balanceadores de carga de aplicación, las condiciones pueden basarse no solo en el origen/destino de las peticiones, sino también en el contenido de esas peticiones.

Los listeners de aplicación pueden ser *HTTP* o *HTTPS*. En el caso de los segundos, se debe asignar un certificado *SSL*. Este tipo de configuración es usual ya que el manejo del cifrado de tráfico *HTTPS* es bastante costoso por lo que se deja esa tarea al balanceador de carga mientras que las instancias se encargan de servir las peticiones web.

Para resumir y recoger los conceptos previamente mencionados, en la *Figura 3-5* se ilustra una configuración de balanceo de carga con 2 *listeners* con al menos una regla, que definen a qué grupos de destino se deben redirigir las peticiones. Para cada grupo de destino existe un chequeo de estado para así evitar situaciones en las que se redirijan mensajes a nodos que no están disponibles.

[21] Una cookie es una pequeña información enviada por un sitio web y almacenada en el navegador web del cliente, de manera que el sitio web puede consultar la actividad previa del navegador.

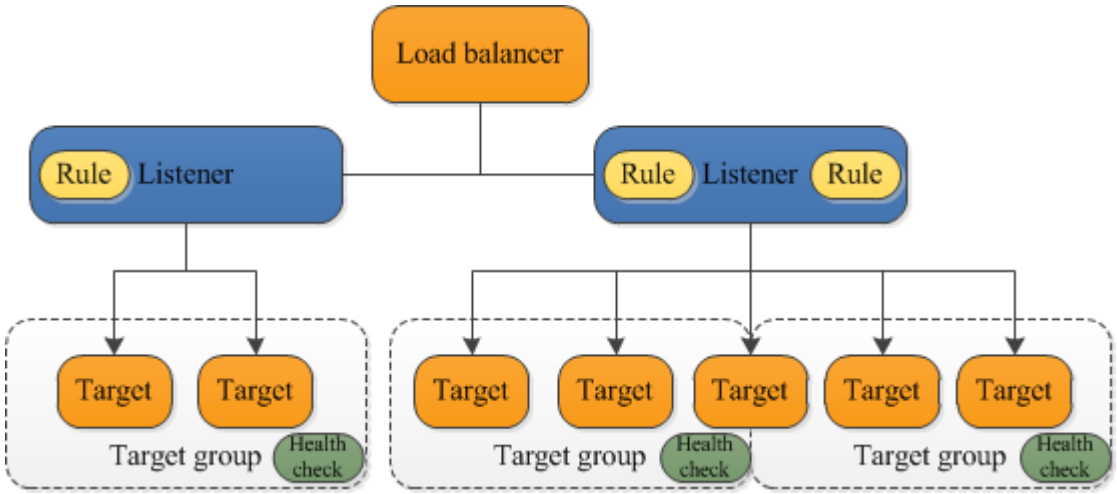


Figura 3-5. Configuración de balanceo de carga. [32]

3.2.4 Escalado automático

Como siguiente grupo de recursos utilizado en este proyecto se definen los grupos de escalado automático (*Auto Scaling Group* o *ASG*).

Este conjunto de recursos es el encargado de incrementar o disminuir el número de instancias que ejecutan un determinado servicio. Para tomar esas decisiones se definen políticas de escalado que se basan en diversas métricas.

El objetivo de esta asignación dinámica es ofrecer un rendimiento de la aplicación correcto, pero al menor coste posible.

3.2.4.1 Auto Scaling Group

El recurso principal que se define en los grupos de escalado automático es precisamente un *Auto Scaling Group*. Define un conjunto de instancias *EC2* que comparten la misma configuración y, por tanto, cada una de las instancias de forma individual no son imprescindibles e incluso son intercambiables entre sí.

Las propiedades más relevantes que define el *Auto Scaling Group* son el número de instancias deseado por defecto, el número mínimo y máximo de instancias o la forma y orden en que nuevas instancias se crean y destruyen [33].

También es posible configurar notificaciones que alerten al administrador de las acciones que se están realizando de forma automática.

3.2.4.2 Launch Configuration

En ocasiones es suficiente con definir en un *Auto Scaling Group* qué AMI se desea lanzar. En otras ocasiones, la configuración a lanzar es más compleja que una simple instancia con valores por defecto. Para ese segundo caso, se pueden utilizar los *Launch Configuration*.

En ellos se puede definir como se quiere personalizar una determinada *AMI*: Tamaño, volúmenes *EBS* que asignar, metadatos…

3.2.4.3 Scaling Policy

Para tomar las decisiones automáticas de cual es el número adecuado de instancias que debe haber en un determinado momento se utilizan políticas de escalado basadas en métricas. Estas métricas pueden ser de tráfico cursado o de uso de recursos en las instancias [34].

Cuando se utilizan políticas de tráfico, el grupo de auto escalado obtiene información por parte del balanceador de carga. Por otra parte, a mayor volumen de tráfico, mayor será el consumo de *CPU* o memoria de las instancias. También se pueden definir métricas como el uso medio de *CPU* y escalar en base a esos datos.

3.2.5 Route 53

El servicio de *DNS* de *AWS* se denomina *Route 53*. Principalmente permite registrar nuevos nombres de dominio y ofrece servicio de traducción de nombres a direcciones *IP* (*DNS*).

3.2.5.1 Registro DNS

Existe un recurso que se denomina registro *DNS*. No es más que una entrada que asigna una determinada *IP* o alias a un nombre de dominio. Sus propiedades más relevantes son el tipo de registro (*A*, *CNAME*, *MX*…) sus valores y el dominio al que se refiere.

3.2.6 Gestor de certificados SSL

AWS dispone de un servicio de generación y validación de certificados *SSL* con los que poder autenticar y cifrar conexiones *HTTPS*. Cada certificado generado debe asegurar que la identidad del solicitante es correcta. De esa forma se garantiza la autenticación, es decir, que el solicitante es quien dice ser. Las validaciones se basan en el paso de información a *AWS* desde un origen al que solo el solicitante tenga acceso.

En *AWS*, esta validación se puede hacer de dos formas. Por un lado, se envía un correo electrónico a las direcciones de administración asociadas al dominio a validar (por ejemplo admin@dominio.com). El solicitante recibe un enlace al que debe acceder para finalizar la validación.

Otro tipo de validación se hace a través del registro *DNS*. El solicitante debe añadir una nueva entrada en el registro *DNS* del dominio en cuestión creando un alias que apunta a subdominios de *AWS* ficticios. Gracias a los mecanismos de propagación de los registros *DNS*, *Route 53* recibe esa nueva entrada y puede validar así el certificado para ese dominio. Debido a los tiempos propagación de la información de los servidores *DNS*, la validación de los certificados siguiendo este método puede llegar a ser de hasta varias horas.

Tabla 3-1 Recursos AWS utilizados.

Servicio	Tipo	Recurso	Descripción
EC2	Instancias	AMI	Imagen de máquina virtual.
		Instancia (VM)	Servidor virtual.
	Redes y seguridad	IP elástica	Dirección IPv4 pública y reservada.
		VPC	Partición de la cloud para uso privado.
		Subred	Subdivisión de una VPC.
		Grupo de seguridad	Cortafuegos virtual para control de tráfico.
	Almacenamiento	Volumen EBS	Dispositivo de almacenamiento de bloques.
		Sistema de Ficheros Elástico	Sistema de ficheros compartido compatible con NFS.
		MountTarget	
RDS	Instancias	Instancia de base de datos	Servidor virtual que sirve una base de datos.
	Clúster	Clúster instancias de base de datos	Grupo de instancias de base de datos.
Balanceo de carga		Balanceador de carga de nivel de aplicación	Nodos para redistribución de tráfico.
		Listener	Entidades para la definición de tráfico y reglas de enrutamiento.
		Grupo de destino	Nodos que reciben tráfico de los balanceadores.
Escalado automático de recursos		Grupo de escalado automático	Define conjuntos de instancias que pueden actuar en grupo.
		Launch Configuration	Elemento a escalar formado por una imagen (AMI) y su configuración inicial.
		Política de escalado	Reglas establecidas para tomar decisiones de escalado.
Route 53	Servicio DNS y dominios	Registro DNS	Entrada en el servicio DNS para asignar una traducción de nombre.
ACM	Gestión de certificados	Certificado SSL	Generación y validación de un certificado SSL.

3.3 CloudFormation

Si bien todos los recursos mencionados en la sección anterior pueden crearse de forma manual a través del portal de *AWS*, no parece muy conveniente hacerlo de esa manera. Existe un servicio que, mediante la creación de plantillas (*templates*) que empaquetan los recursos que se quieren lanzar, pueden gestionar los recursos de forma global y automática. En el caso de *AWS*, este servicio se denomina *AWS CloudFormation*.

El desarrollo de estas templates puede parecer conceptualmente simple, pero en la práctica, la variedad de recursos y configuraciones hacen que no lo sea tanto. Sin ir más lejos, el manual de referencia de *CloudFormation* tiene cerca de 2500 páginas de extensión [35].

Como formato de datos para el modelado de recursos se utiliza *JSON* (*JavaScript Object Notation* https://www.json.org/) o *YAML* (*YAML Ain't Markup Language*[22] http://yaml.org/). Son dos lenguajes de modelado y serialización de datos que presentan in buen balance entre facilidad de lectura/escritura por humanos, así como facilidad para el parseo por máquinas.

En la *Tabla 3-2* se muestra un sencillo ejemplo de la misma estructura de datos representada en los dos lenguajes.

Tabla 3-2. Representación de datos. JSON y YAML.

```json
[
  {
    "tipo": "perro",
    "nombre": "Rocky",
    "edad": 3
  },
  {
    "tipo": "pez",
    "nombre": "Nemo",
    "edad": 1
  }
]
```

```yaml
---

# Lista de mascotas
- tipo: perro
  nombre: Rocky
  edad: 3
- tipo: pez
  nombre: Nemo
  edad: 1
```

También es posible utilizar *AWS CloudFormation Designer* (*Figura 3-6*), una herramienta visual, que como es habitual en este tipo de servicios, no ofrece una funcionalidad completa, pero es adecuado para dar los primeros pasos e introducirse en *CloudFormation*.

[22] Es común encontrar acrónimos recursivos en el campo de la programación. Uno de los más conocidos es el de GNU: *"GNU's Not Unix"*.

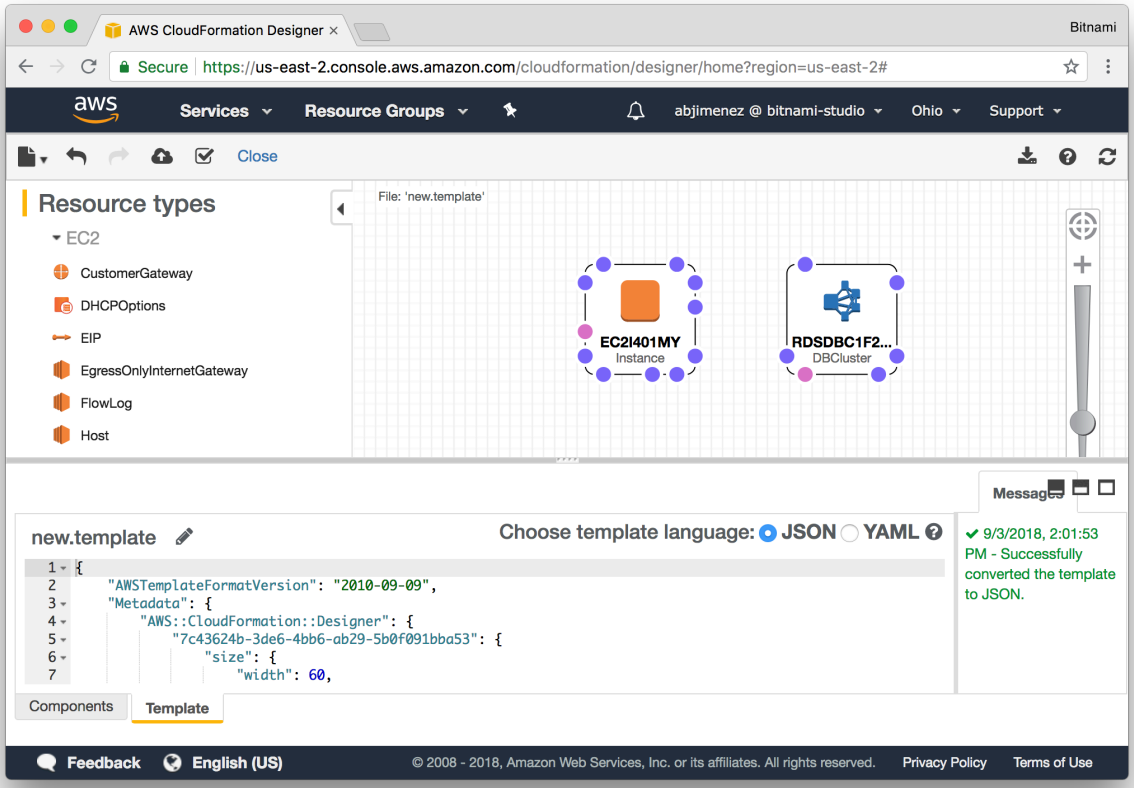


Figura 3-6. Interfaz de CloudFormation Designer.

3.3.1 Características de CloudFormation

CloudFormation pretende ofrecer una solución en la que poder definir la infraestructura de un despliegue como si fuese código, de forma que este código se pueda incluir en un repositorio de control de versiones y se pueda compartir de manera cómoda para así facilitar la colaboración en el desarrollo de forma grupal. Otra gran ventaja que ofrece el definir la infraestructura como código es la reproducibilidad que esto permite. Se pueden desplegar los mismos recursos de forma cómoda tantas veces como se quiera.

Algunas de las características de *CloudFormation* son [36]:

- Despliegue automático de los recursos sin intervención manual. Esto evita errores humanos que se puedan dar.
- Visualización de cambios que se puedan dar en un despliegue o actualización.
- Administración de dependencias entre recursos para garantizar el orden correcto cuando se hace despliegue.
- Portabilidad. Se puede lanzar la misma infraestructura en diversas regiones o cuentas, simplemente hay que usar las mismas plantillas.
- Extensibilidad. Se pueden crear conjuntos de recursos (*stacks*) que actúan como un elemento único. De esa forma se pueden anidar recursos y reutilizarlos por medio de importaciones.

3.3.2 Estructura de las plantillas

Cada una de las plantillas tiene una serie de secciones, siendo la sección de recursos la única obligatoria [37]. Estas secciones describen como se interactúa con las plantillas y qué recursos se despliegan. Las secciones se presentan en la *Tabla 3-3* de forma esquemática. Por mencionar algunas de las más relevantes:

- *Parameters.*
  Definen una serie de variables de entrada que serán especificadas en tiempo de ejecución para parametrizar y personalizar una *stack*.
- *Conditions.*
  Definen funciones condicionales en base a parámetros. Utilizando estas condiciones, se puede elegir si un recurso debe ser creado o no.
- *Resources.*
  La sección obligatoria. Define los recursos que se van a desplegar y como estos se configuran.
- *Outputs.*
  Definen variables de salida que el usuario (u otra *stack*) reciben. El caso más común es definir como un *output* la dirección IP de una instancia lanzada.

Tabla 3-3. Estructura básica de las plantillas CloudFormation. [37]

```
{
  "AWSTemplateFormatVersion" : "version
date",

  "Description" : "JSON string",

  "Metadata" : {
    template metadata
  },

  "Parameters" : {
    set of parameters
  },

  "Mappings" : {
    set of mappings
  },

  "Conditions" : {
    set of conditions
  },

  "Transform" : {
    set of transforms
  },

  "Resources" : {
    set of resources
  },
```

```
---
AWSTemplateFormatVersion: "version date"

Description:
  String

Metadata:
  template metadata

Parameters:
  set of parameters

Mappings:
  set of mappings

Conditions:
  set of conditions

Transform:
  set of transforms

Resources:
  set of resources

Outputs:
  set of outputs
```

```
  "Outputs" : {
    set of outputs
  }
}
```

3.3.3 Señalización

Con el servicio *CloudFormation* no solo se definen los recursos que se quieren desplegar por medio de plantillas, sino que además se permite lanzar y eliminar estos. Para respetar un orden en la creación de recursos se utiliza señalización entre los recursos y *CloudFormation*.

3.3.4 Alternativas globales: Terraform

Cada *cloud*[23] define un sistema propio de plantillas mediante el cual se pueden desplegar recursos y que, aunque los conceptos son los mismos, la forma de definirse y configurarse es completamente distinta, tanto por las plantillas en sí como por las herramientas que se utilizan para desplegar la infraestructura, de forma que se hace complejo el uso de escenarios de infraestructura para el despliegue en múltiples *clouds*.

La compañía *HashiCorp* ofrece un sistema de *templates* global denominado *Terraform* que pretende ofrecer una solución a este problema. Si bien es cierto que el conjunto de configuraciones posibles no es tan amplio como el que ofrecería el sistema nativo, para un amplio número de escenarios, es una alternativa válida.

Para los ficheros *Terraform* se utiliza el lenguaje *HCL* (*HashiCorp Configuration Language*), que pretende combinar las ventajas de los lenguajes *JSON* y *YAML* [38].

Mediante la utilización de una herramienta de línea de comandos, estos ficheros son leídos y se puede desplegar infraestructura en diferentes *clouds*. Para ilustrar la funcionalidad de *Terraform*, en la *Tabla 3-4* se muestra como se puede definir una instancia de *AWS*.

Tabla 3-4. Definición de instancia AWS mediante Terraform.

```
provider "aws" {
  # access_key = "ACCESS_KEY_HERE"
  # secret_key = "SECRET_KEY_HERE"
  region     = "us-east-1"
}

resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
}
```

Para ver ejemplos más completos del uso de *Terraform*, se puede consultar el repositorio público de código https://github.com/bitnami/oci-multi-tier[24]. En él se definen soluciones de múltiples instancias que *Bitnami* mantiene, 4 por el momento. (*Figura 3-7*).

[23] O casi todas. Por ejemplo *Oracle Cloud Infrastructure* aún no tiene un sistema de plantillas para definir recursos.
[24] La creación soluciones basadas en plantillas Terraform para la cloud OCI ha sido un proyecto paralelo que he podido liderar.

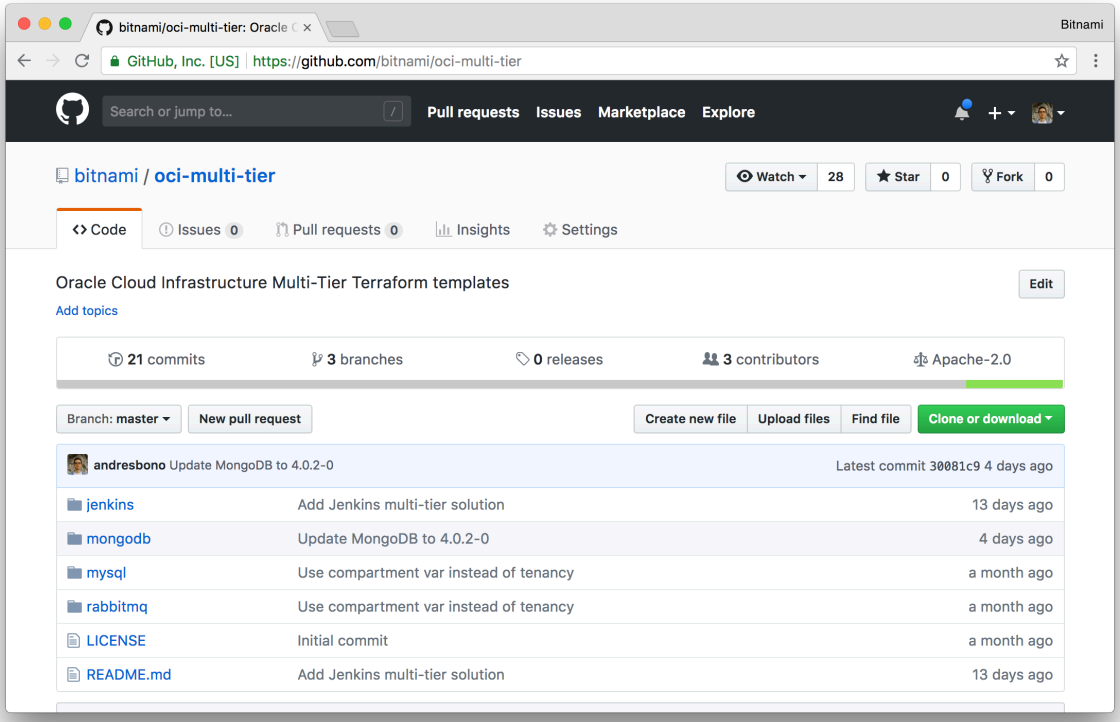


Figura 3-7. Repositorio de plantillas Terraform para la cloud OCI.

3.4 Quick Starts

3.4.1 Qué son las soluciones Quick Start

Las *Quick Starts* son las soluciones de referencia para el despliegue de servicios y recursos de la cloud *AWS*. Lanzan, configuran y ejecutan múltiples elementos de *cloud* (instancias, redes, volúmenes de almacenamiento...) de forma totalmente automática a partir de un solo clic, eliminando cientos de pasos manuales que serían necesarios para configurar cada uno de estos elementos. Esto es posible gracias a las plantillas *CloudFormation*.

Las soluciones clasificadas como *Quick Starts* son diseñadas por arquitectos de *AWS* o por sus socios (entre los que se encuentra *Bitnami*) y se consideran "*Automated, gold-standard deployments in the AWS Cloud*" [39] ya que siguen las buenas prácticas en cuanto a topologías y seguridad que *AWS* exige.

Si se observa el catálogo de soluciones que se incluyen, se puede apreciar la amplia variedad de aplicaciones que incluye. Existen desde clústeres de *MongoDB*[25] con replicación de base de datos y mecanismos de recuperación ante fallos hasta clústeres de *Kubernetes*. Son aplicaciones que no son para nada triviales de configurar. Cualquier experto en estas tecnologías puede confirmar que la cantidad de tiempo y dinero que serían necesarios emplear para montar las infraestructuras para este tipo de aplicaciones desde cero son considerables.

[25] MongoDB es un gestor de bases de datos NoSQL de código abierto.

Otra de los objetivos que *AWS* quiere conseguir con las *Quick Starts* es hacer disponible estas soluciones como software libre, de forma que los usuarios de la *cloud* puedan leer y aprender cómo realizar este tipo de despliegues de forma automática.

AWS no es la única cloud que dispone de este tipo de soluciones de referencia. *Azure*, por ejemplo, también dispone de ellas en su portal: https://azure.microsoft.com/en-us/resources/templates/.

3.4.2 Requisitos de las soluciones Quick Start

3.4.2.1 Proceso de solicitud

Para la creación de nuevas soluciones con la calificación de *Quick Starts* es necesario seguir un proceso de solicitud que se describe en su guía de contribuidor: https://aws-quickstart.github.io/ [40].

Para empezar, hay que presentar la idea incluyendo cierta información como la descripción del proyecto, la organización del código, las personas de contacto las licencias bajo las que se distribuye el software que la solución incluye y también la firma de un acuerdo legal. Esta primera parte del proceso se agiliza bastante en el caso de ser socio de *AWS* y tener un agente de contacto.

Tras la aceptación del proyecto, el siguiente paso es preparar la solución, testearla y crear un documento llamado "*deployment guide*" donde se detallan los pasos necesarios para lanzar la solución, así como las características con las que cuenta la *Quick Start*. Este documento está destinado a los usuarios finales por lo que debe ser tan completo como sea posible.

Una vez estos tres elementos están listos, se envía al equipo de *AWS* encargado del mantenimiento de las *Quick Starts* y lo integran en el repositorio de código *GitHub*[26] para realizar una revisión de código y aportar ideas y mejoras que aplicar en la solución.

Para finalizar, cuando la solución se considera apta para ser publicada, se coordinan anuncios de marketing para que el lanzamiento tenga un impacto considerable.

Una vez la solución es finalmente publicada, debe ser mantenida. Esto significa que se debe ofrecer soporte a los usuarios finales en el caso de que estos tengan dudas o encuentren errores en la solución. En el caso de que aparezcan nuevas versiones del código de la aplicación o ante incidentes de seguridad, las *Quick Starts* deben ser actualizadas.

3.4.2.2 Requisitos técnicos

Existen una serie de requisitos técnicos que las soluciones deben cumplir para ser aptas para su publicación, están bien definidos y se enumeran en la guía del contribuidor [40]. Estos requisitos se pueden clasificar en varios grupos recogidos en la *Tabla 3-5*: topológicos, de configuración de red, buenas prácticas y seguridad.

[26] GitHub es una plataforma de desarrollo colaborativo que almacena proyectos de código libre utilizando el sistema de control de versiones Git.

Tabla 3-5 Requisitos técnicos de las soluciones Quick Start

Tipos	Requisitos
Topología	Arquitectura con múltiples zonas de disponibilidad. Soporte en la mayoría de regiones de AWS. *
Configuración de red	Opciones de despliegue en nueva y existente VPC. Las máquinas virtuales del producto deben estar en redes privadas. Pasarelas NAT[27] para tráfico saliente de las subredes privadas.
Buenas prácticas	No utilizar AMIs pre construidas. * No utilizar referencias estáticas a AMIs. Utilizar nombres descriptivos para los recursos. No requerir recursos ya existentes fuera de la solución.
Seguridad	Evitar acceso como usuario administrador a las máquinas. Crear reglas de seguridad siguiendo el principio del mínimo privilegio. No incluir software desconocido en la solución. No incluir contraseñas en el código. No incluir ningún tipo de datos sensibles en la *user-data* de las instancias. No usar como el CIDR 0.0.0.0/0 como origen del tráfico para la gestión de la solución.

La amplia mayoría de estos requisitos son cumplidos por la solución *WordPress High Availability*. Sin embargo, los marcados con un asterisco (*) no se cumplen. En el capítulo 8 se explica la justificación de este hecho.

3.5 Otras herramientas de desarrollo

3.5.1 Herramienta por línea de comandos

AWS pone a la disposición de los usuarios una herramienta de software libre (https://github.com/aws/aws-cli) por línea de comandos escrita en *Python*[28] para la gestión de los

[27] NAT aquí se refiere a la técnica de traducción de direcciones de red (o enmascaramiento IP) que se utiliza como solución al finito número de direcciones IP que existen. Este mecanismo también aporta una capa extra de seguridad.
[28] Lenguaje de programación interpretado orientado a objetos.

servicios. Soporta una gran variedad de comandos para cada uno de los productos de *AWS*. La salida resultante de la ejecución de comandos suele presentarse en formato *JSON*.

Como ejemplo, en la *Figura 3-8*, se ilustra cuál sería el comando para obtener una lista de las imágenes publicadas por *Bitnami* (argumento *owner*) que coincidan con un determinado nombre, en este caso, el nombre para las imágenes de la solución *WordPress High Availability*.

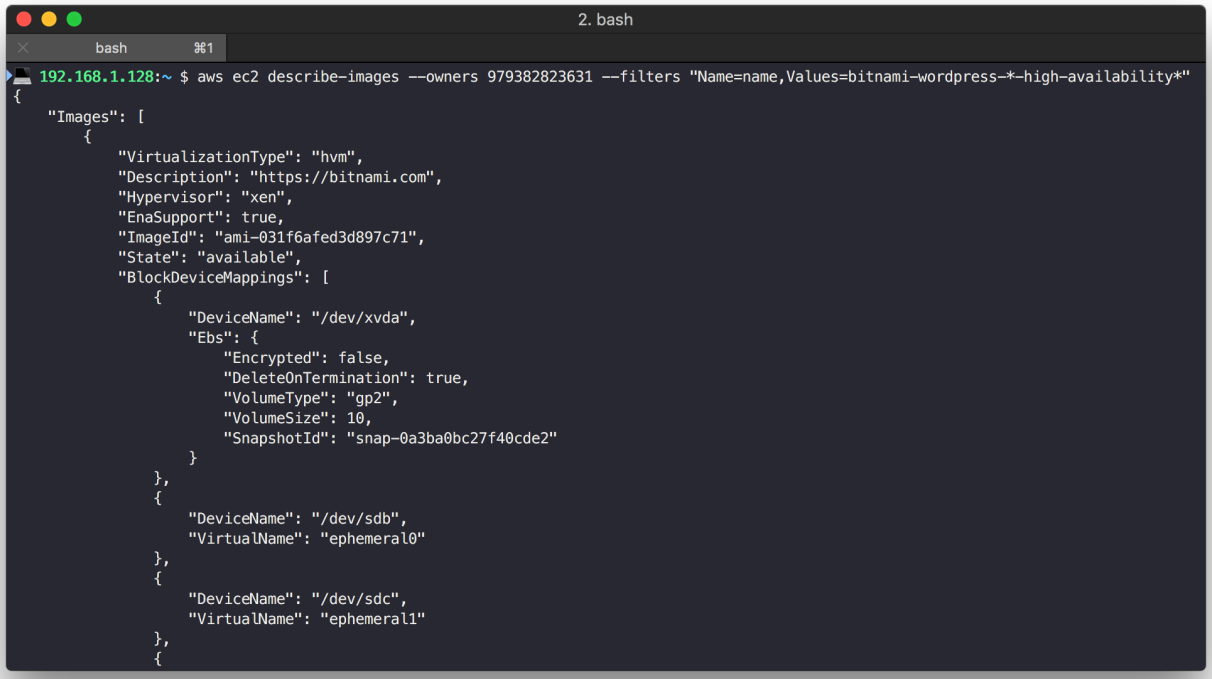


Figura 3-8. Ejecución de comando describe-images con aws-cli.

3.5.2 Kits de desarrollo

Existen kits de desarrollo o *SDKs* (*Software Development Kit*) para los principales lenguajes de programación (*Figura 3-9*). Con ellos, es posible acceder a los servicios de *AWS* mediante una *API*[29] adaptada al lenguaje de programación que se esté utilizando para desarrollar nuevos programas que se integran con *AWS*.

[29] Una API (Application Programming Interface) es una interfaz de comunicación bien definida por medio de métodos o procedimientos para la comunicación con un servicio.

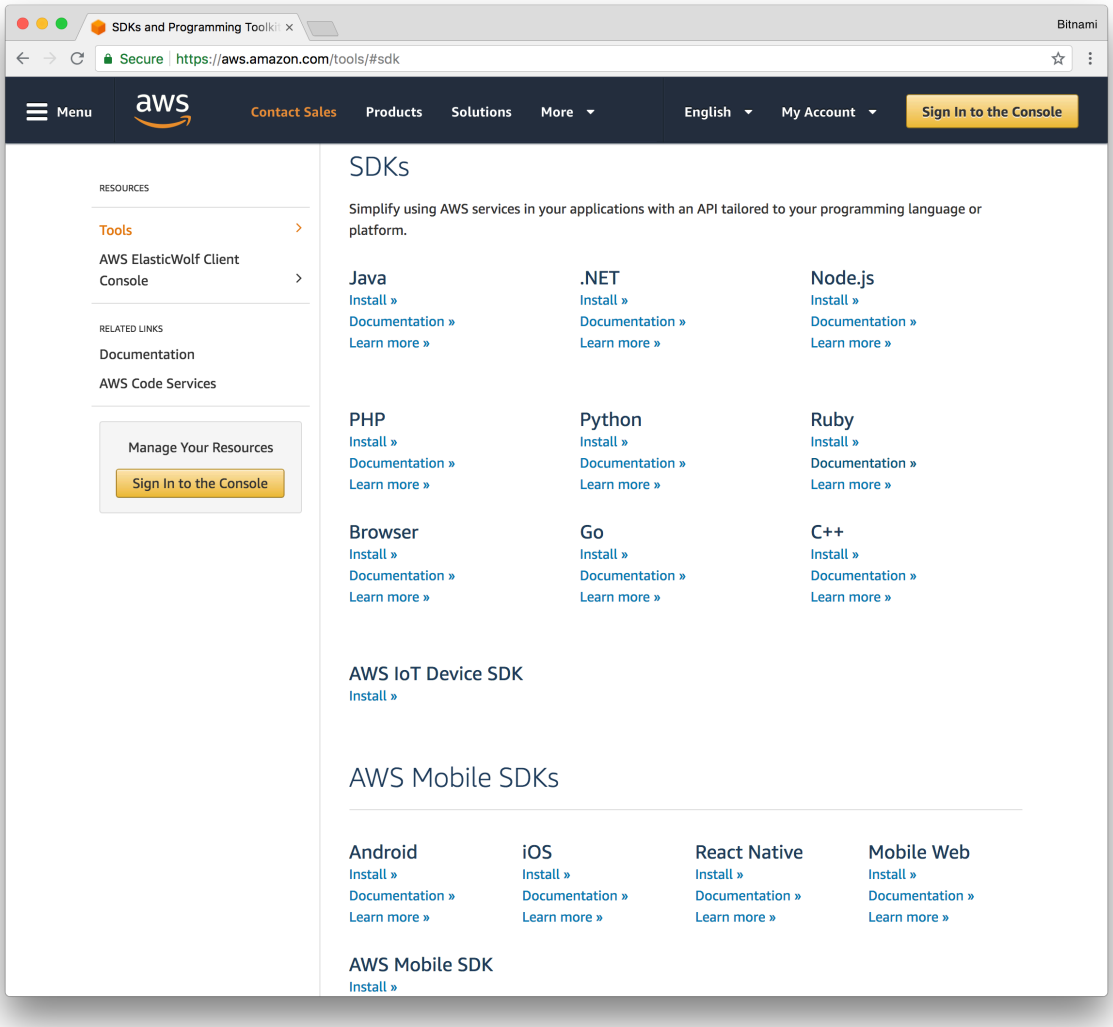


Figura 3-9. SDKs de AWS para multitud de lenguajes de programación. [41]

4 ARQUITECTURA DE LA APLICACIÓN

En la sección 3.4 se definen qué son las *Quick Starts* y qué requisitos deben cumplir para considerarse las soluciones de referencia de *AWS*. En la sección 3.2, *Tabla 3-1* se muestran los recursos utilizados en esta solución.

En este capítulo se pretende mostrar la arquitectura elegida para el producto *WordPress High Availability* que cumple con dichos requisitos y cómo los recursos enumerados previamente interactúan entre sí.

4.1 Migrar aplicaciones tradicionales a la cloud

Como demuestran los datos de la sección 2.4 sobre la comparación de servicios *cloud*, las empresas saben que el futuro de los servicios pasa por la utilización de infraestructura en la *cloud*. Para sacar el máximo partido de las ventajas de la nube, las aplicaciones deben haberse diseñado siguiendo una serie de estándares (como los enumerados en https://12factor.net/) que hagan que la aplicación pueda escalar sin problemas estableciendo numerosas divisiones entre cada una de las tareas que la aplicación tiene que realizar, de forma que cada uno de estos procesos (también denominados micro servicios) puedan ser gestionados de forma separada.

A estas aplicaciones se las denominan Aplicaciones *Cloud* Nativas o *Cloud Native Applications.* Pero la realidad es que la inmensa mayoría de aplicaciones no siguen estos estándares de diseño ya que cuando fueron desarrolladas, fueron concebidas para ejecutarse en servidores monolíticos[30].

Por el simple hecho de mover los servicios de una máquina monolítica a una instancia *EC2* en la nube ya se están obteniendo beneficios (como ahorro de tareas de mantenimiento), pero serían muchos más si la aplicación tomase la forma de una topología más distribuida.

Ante esta situación, se aportan algunas soluciones a los retos que hay que resolver para hacer de una aplicación tradicional, como es el caso de *WordPress*, un servicio web adaptado a la *cloud*.

[30] Un servidor monolítico se define como una única máquina que realiza todas las operaciones para dar servicio a una aplicación y que almacena todos los datos de esta.

4.1.1 Separación de los servicios en diferentes máquinas

En *WordPress* existe un único servicio de cara al usuario: un servicio web también denominado *front-end*. Normalmente se sirve a través del puerto 80 (o 443 si te utiliza *HTTPS*) utilizando un servidor web como *Apache* o *Nginx*[31].

Sin embargo, para su funcionamiento, *WordPress* hace uso de una base de datos *SQL*, comúnmente *MySQL*, donde se almacena información de la aplicación como son los usuarios registrados o las publicaciones realizadas. Su puerto *TCP* por defecto es el 3306. A esta parte de la aplicación se le suele denominar *back-end*.

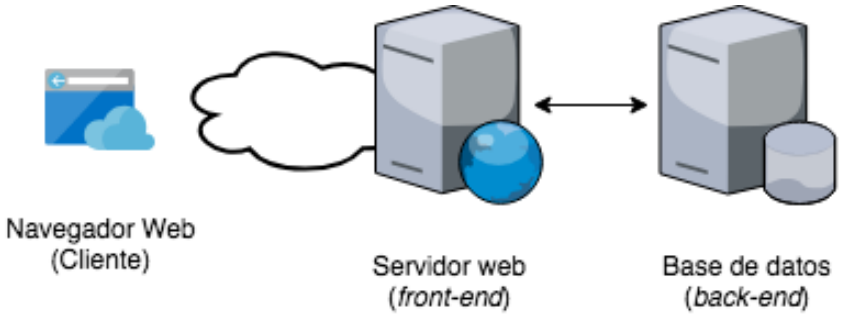


Figura 4-1. Separación del servicio web y la base de datos.

Con una separación como la explicada y representada en *Figura 4-1*, se consigue una administración de servicios de forma independiente. Tiene algunos beneficios directos como el poder incrementar el tamaño de las instancias de forma personalizada si alguno de los servicios lo requiere o un aumento del rendimiento de los servicios al tener más capacidad de computación (dos máquinas en lugar de una).

La aplicación *WordPress* está preparada para realizar esta separación. En su fichero de configuración (*wp-config.php*) se puede especificar un destino de la base de datos (una dirección *IP*) que sea distinta de la dirección local por medio de la variable *DB_HOST* (*Tabla 4-1*).

Tabla 4-1. Sección de la configuración de base de datos en WordPress

```
// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define('DB_NAME', 'bitnami_wordpress');

/** MySQL database username */
define('DB_USER', 'bn_wordpress');

/** MySQL database password */
define('DB_PASSWORD', '');

/** MySQL hostname */
define('DB_HOST', 'mariadb:3306');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');
```

[31] Se trata de dos servidores web HTTP de software libre.

```
/** The Database Collate type. Don't change this if in doubt. */
define('DB_COLLATE', '');
```

También es importante establecer esta separación ya que es posible que se necesite aumentar no solo el tamaño de las instancias, sino también el número. La separación entre servicios ayuda a poder compartirlos entre varias máquinas. Esto quiere decir que varios nodos *front-end* podrían acceder a un mismo servicio *back-end* si fuese necesario (*Figura 2-1*).

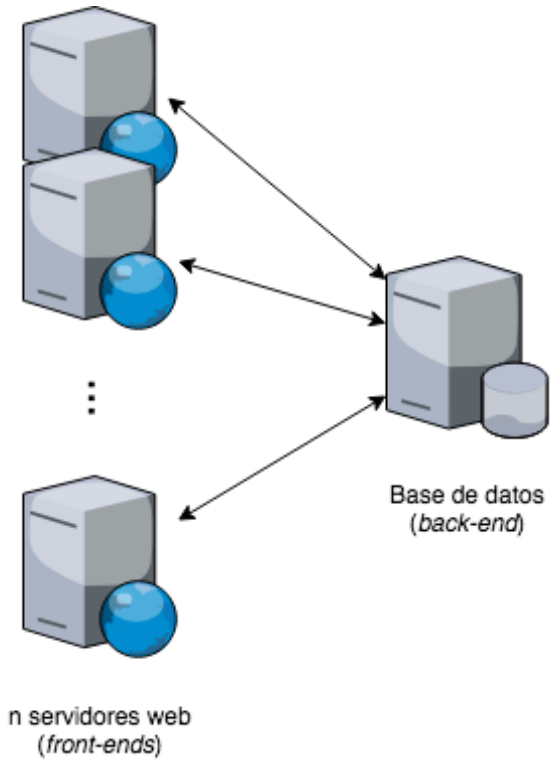


Figura 4-2. Configuración con n servidores web accediendo a una misma base de datos.

4.1.2 Utilización de imágenes de instancias

Para esta aplicación se define una imagen de máquina virtual (*AMI*) que incluye todo lo necesario para poder hacer funcionar el servicio web. Estos componentes se enumeran en el capítulo 6. Al hacer uso de imágenes o *snapshots*[32], se consigue que la puesta en marcha de las máquinas sea muy rápida y eficiente, aportando una gran flexibilidad.

Tomando como ejemplo la *AMI* de la región *us-east-1* de *WordPress High Availability* (versión más reciente), se puede analizar cuanto tiempo tarda en iniciarse. La *Tabla 4-2* recoge un script que puede utilizarse para calcular el tiempo que una instancia tarda en estar totalmente disponible. Aunque este tiempo dependerá de la carga del servicio *EC2* en ese momento, sirve para definir el orden de magnitud del que se está hablando. La *Figura 4-3* muestra la ejecución del script, invirtiendo menos de 5 minutos en desplegar e inicializar completamente la instancia.

El script utiliza la herramienta de línea de comandos de *AWS* (comandos *run-instances* y *describe-instance-status*) para consultar el estado de la instancia lanzada. Para *parsear*[33] la salida

[32] El término snapshot se utiliza comunmente para referirse a un archivo que contiene toda la información de un determinado sistema en un instante de tiempo concreto.
[33] Analizar una cadena de texto que tiene una sintaxis y formato conocidos para poder acceder a los campos del texto que interesan.

del comando, que como se ha comentado suele obtenerse en formato *JSON*, se utiliza la herramienta *jq*. Se trata de un procesador de *JSON* ligero muy útil para este tipo de situaciones en las que se quiere extraer información para utilizarla en un script.

Tabla 4-2. Shell script que lanza una instancia EC2 y espera hasta su inicialización completa.

```
#!/bin/bash

AMI="ami-074b4f71c0e2efe5b"

INSTANCE_ID="$(
  aws ec2 run-instances \
    --image-id $AMI \
    --count 1 \
    --instance-type t2.micro \
    --key-name abjimenez \
    --security-group-ids sg-24b5ad41 |
  jq --raw-output '.Instances | .[] | .InstanceId'
)"

echo "Instance ID: >$INSTANCE_ID<"

sleep 30
while true; do
  aws ec2 describe-instance-status --instance-id "$INSTANCE_ID" |
  jq --raw-output '.InstanceStatuses | .[] |
.SystemStatus.Status,.InstanceStatus.Status' |
  grep -v ok || break
  sleep 1
done

echo "Instance initialized"
```

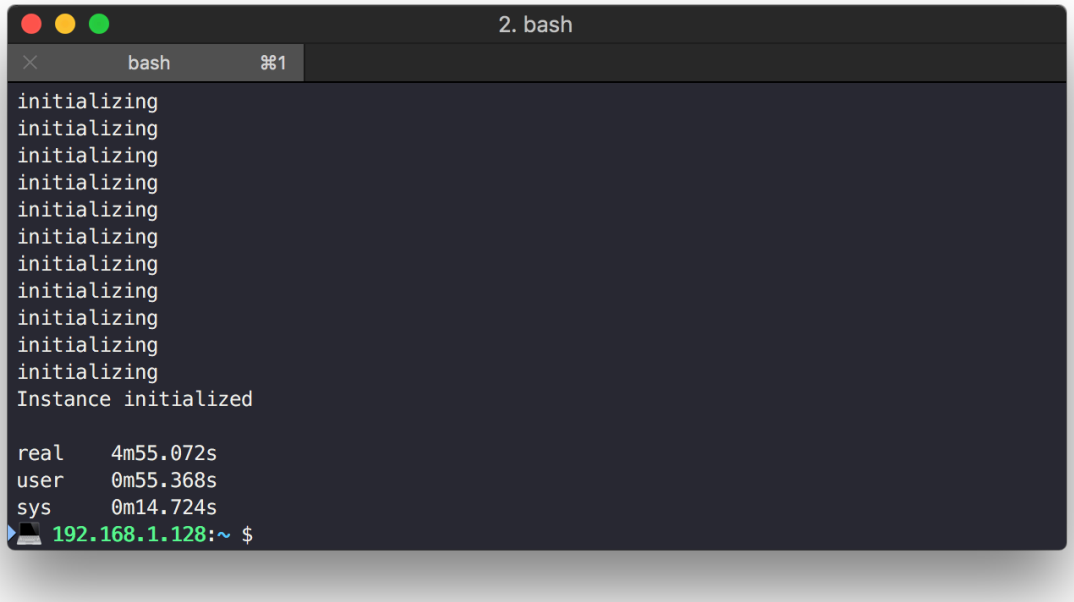


Figura 4-3. Tiempo de ejecución de Shell script lanzando instancia EC2.

Otra de las características que aporta el poder definir *AMIs* es la reproducibilidad que se consigue con ellas. El contenido de 2 instancias que utilizan la misma *AMI* debe ser prácticamente idéntico y su funcionamiento debe diferir.

Las dos características mencionadas (rapidez de inicio y reproducibilidad) son claves para poder hacer uso de los *Auto Scaling Groups*.

4.1.3 Compartición de ficheros

Puesto que se desea tener múltiples máquinas para asegurar la alta disponibilidad, hay que establecer un mecanismo mediante el cual cada una de las instancias pueda acceder a ficheros comunes incluso si estos los ha escrito otra instancia del servicio, ya que idealmente, cada una de las instancias por separado no debe ser imprescindible para ofrecer el servicio.

En el caso de *WordPress*, los ficheros comunes que se deben compartir son:

- El fichero de configuración de la aplicación *wp-config.php*.
- El directorio *wp-content* que contiene toda la información referida a la aplicación que no es guardada en la base de datos. Algunos subdirectorios son:
    - *THEMES*. Almacena los ficheros relativos a los temas instalados.
    - *PLUGINS*. Almacena los ficheros relativos a los plugins instalados.
    - *UPLOADS*. Directorio relativo a las subidas de archivos en la web. Suele contener, por ejemplo, las imágenes que se adjuntan en los *posts*.

Como solución a este problema, se utiliza el servicio *AWS EFS* (*Elastic File System*) descrito en el apartado *3.2.1.7*. Se utiliza para crear y montar un sistema de ficheros compartido. De esa forma, a ojos de la aplicación, estos ficheros compartidos son ficheros locales.

4.1.3.1 Funcionamiento de EFS

Una vez que se crean los recursos *EFS* (El sistema de ficheros en sí y uno o más *MountTargets* para poder acceder a él desde la subred de la instancia), es necesario montar el sistema de ficheros.

Para ello, se puede utilizar la herramienta *aws-cli* (comandos *describe-file-systems* y *describe-mount-targets*) de forma que se pueda obtener la información necesaria. Los comandos son los mostrados en la *Figura 4-4*. Se puede observar como existe un sistema de ficheros con dos destinos de montaje ya que en *WordPress High Availability* se utilizan dos zonas de disponibilidad con diferentes subredes. La topología de red presente en los ejemplos se describe a continuación:

- Región: *ap-southeast-2* (Sydney).
- Zonas de disponibilidad: *ap-southeast-2a* y *ap-southeast-2b*.
- Subredes: *10.0.0.0/19 y 10.0.32.0/19*.

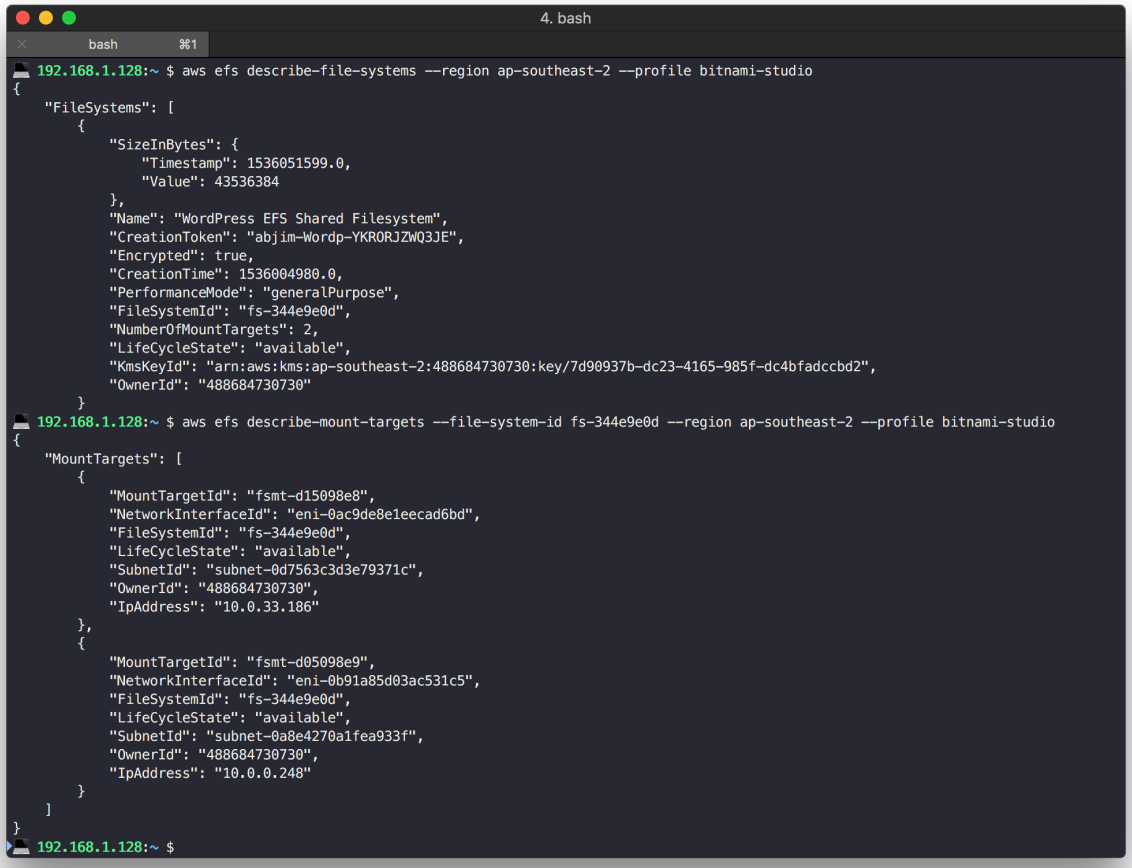


Figura 4-4. Descripción del sistema de ficheros y MountTargets con aws-cli.

Desde las instancias, los *MountTargets* son alcanzables. En la *Figura 4-4* se muestra la dirección *IP* a utilizar, pero es preferible utilizar un nombre de domino identificado por el id del sistema de ficheros que se resuelve mediante *DNS* a una dirección *IP*. En la *Figura 4-5*, se ha accedido mediante *SSH* a dos instancias localizadas en dos zonas de disponibilidad. Utilizando la herramienta *nsloopup* para la realizar solicitudes de traducción de nombres, se puede observar como los nombres de dominio con formato *<zona de disponibilidad>.<id sistema de ficheros>.<región>.amazonaws.com* identifican de forma única a los distintos *MountTargets*. Sin embargo, se puede prescindir de especificar la zona de disponibilidad en el nombre de dominio y gracias a la resolución de *DNS*, la dirección *IP* obtenida es la correspondiente a la zona de disponibilidad en la que se está presente.

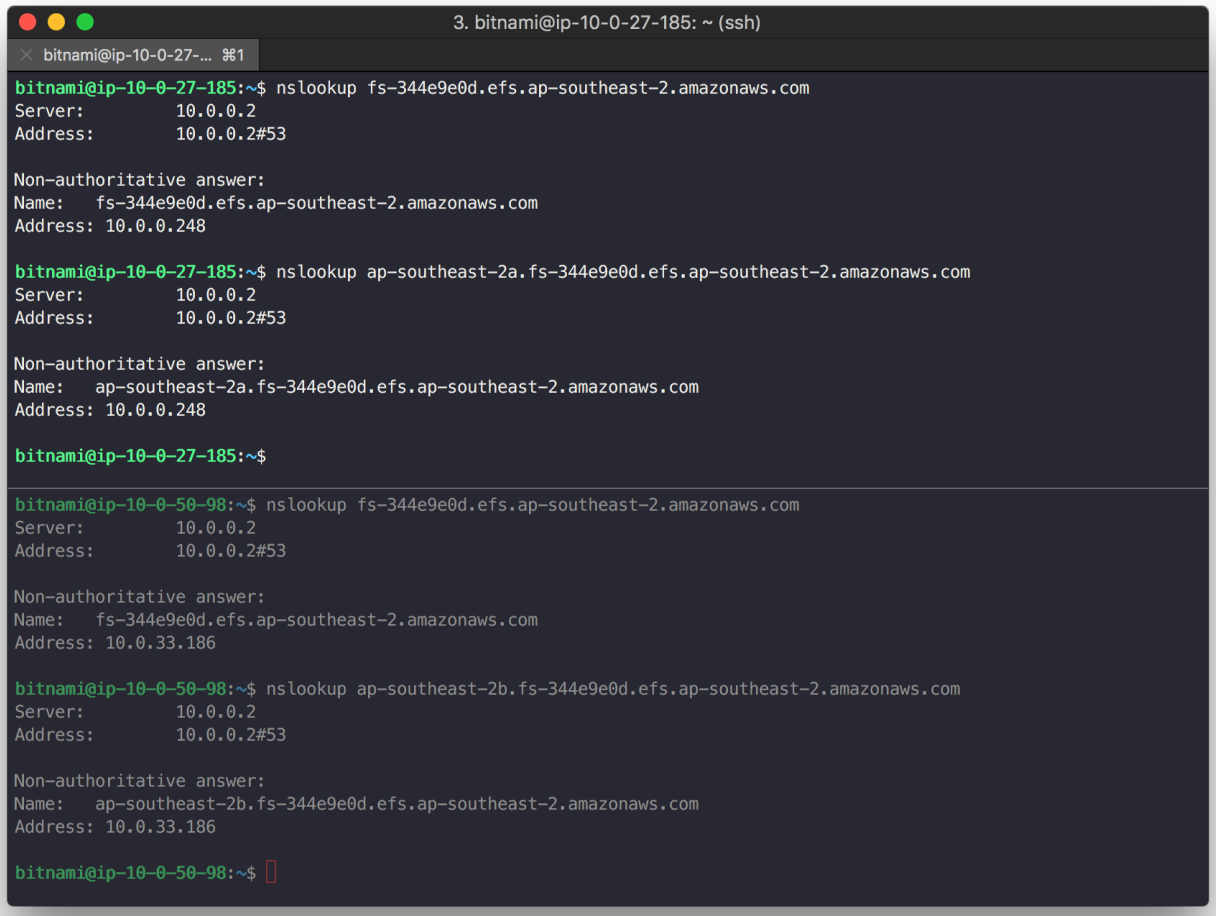


Figura 4-5. Resolución DNS de los MountTargets en distintas zonas de disponibilidad.

Conociendo los puntos de montaje, los sistemas de ficheros se pueden montar con la utilidad *mount* de *Linux* (*Figura 4-6*). Puesto que el protocolo utilizado para la comunicación con el sistema de ficheros compartido es *NFS*, es necesario que el paquete del sistema *nfs-utils* esté presente. Una vez montados, el sistema detecta el sistema de ficheros y este puede ser utilizado.

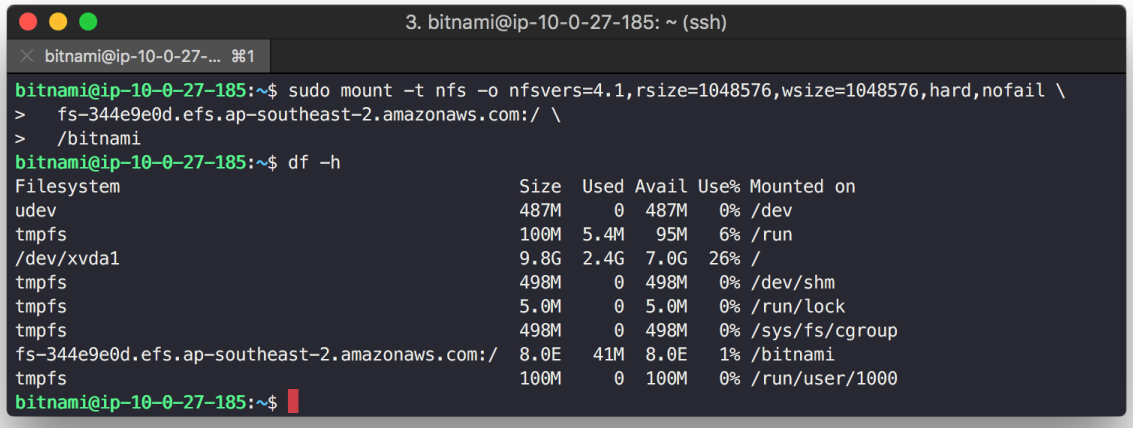


Figura 4-6. Ejecución de comandos mount y df.

Tras esto, solo queda mover los directorios a compartir al nuevo volumen y crear enlaces simbólicos en la ubicación original, de forma que todos estos cambios sean transparentes a la aplicación.

Respecto a la seguridad de este sistema, hay que tener en cuenta que los recursos son desplegados dentro de una *VPC* de *AWS*, por lo que los recursos son accesibles solamente desde dentro de esa red. A esto hay que añadir los grupos de seguridad definidos, que deben establecer reglas específicas para el tráfico *NFS* entre recursos. También es importante destacar que los *MountTargets* son accesibles a través de direcciones *IP* privadas, que no son encaminadas a través de internet.

Con la utilización de una base de datos externa y el uso de *EFS* para compartir los directorios mencionados, se cubren las necesidades de tener compartidos los recursos que puedan crearse a raíz de la utilización de la propia aplicación. Por otro lado, existen ficheros que forman parte del código *PHP* de la aplicación en sí. Estos son independientes en cada instancia, pero no necesitan ser compartidos ya que se trata de ficheros estáticos.

4.1.4 Utilización de zonas de disponibilidad

Para asegurar que el servicio esté siempre disponible incluso cuando se den fallos en la infraestructura de *AWS*, es necesario establecer replicación de recursos. Algunos servicios como *RDS* o *EFS*, si se configuran adecuadamente, ofrecen respaldo en este sentido. Otros como por ejemplo las instancias, por sí solos, no tienen este tipo de respaldo.

Para conseguirlo, se hace uso de grupos de escalado automático que despliegan instancias en múltiples zonas de disponibilidad si se configura para ello (*Tabla 4-3*). Mediante el uso de la propiedad *VPCZoneIdentifier* [42], que contiene una lista de subredes donde se desean desplegar las instancias. Las subredes definidas deben estar en zonas de disponibilidad diferentes.

Tabla 4-3. Extracto de configuración del grupo de auto escalado para soporte de múltiples zonas de disponibilidad.

```
        "WebServerASG": {
            "Type": "AWS::AutoScaling::AutoScalingGroup",
            "Properties": {
                "VPCZoneIdentifier": {
                    "Ref": "WebServerSubnets"
                },
            },
        },
```

Conceptualmente, la utilización de múltiples zonas de disponibilidad se consigue duplicando las subredes definidas y haciendo que los recursos se desplieguen en ellas. En *WordPress High Availability* se puede observar (*Figura 4-7*) como las instancias se despliegan en diferentes zonas de disponibilidad de forma automática (*apsoutheast-2a* y *apsoutheast-2b*)[34]. Es importante ver como los identificadores de instancia son diferentes, puesto que son máquinas independientes, pero parten de la misma *AMI*, puesto que el propósito de estas instancias es el mismo.

[34] En esta vista de la consola de AWS se han filtrado (no se muestran) las instancias bastión que se describen posteriormente.

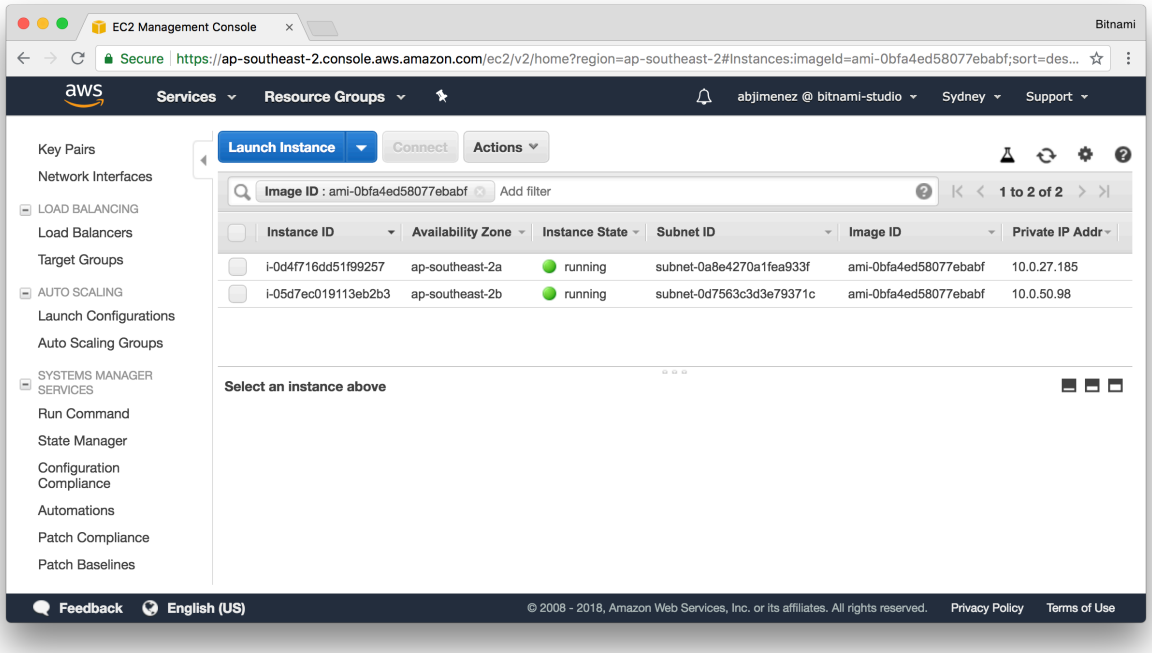


Figura 4-7. Instancias desplegadas en múltiples zonas de disponibilidad.

En *WordPress High Availability*, para la definición de subredes con soporte para múltiples zonas de disponibilidad, se han utilizado las plantillas de referencia para la definición de *VPCs*. El equipo *AWS Quick Starts* mantiene estos ficheros de forma que otros proyectos puedan importarlos y hacer uso de ellos. La topología de red se puede ver en la *Figura 4-8* [43].

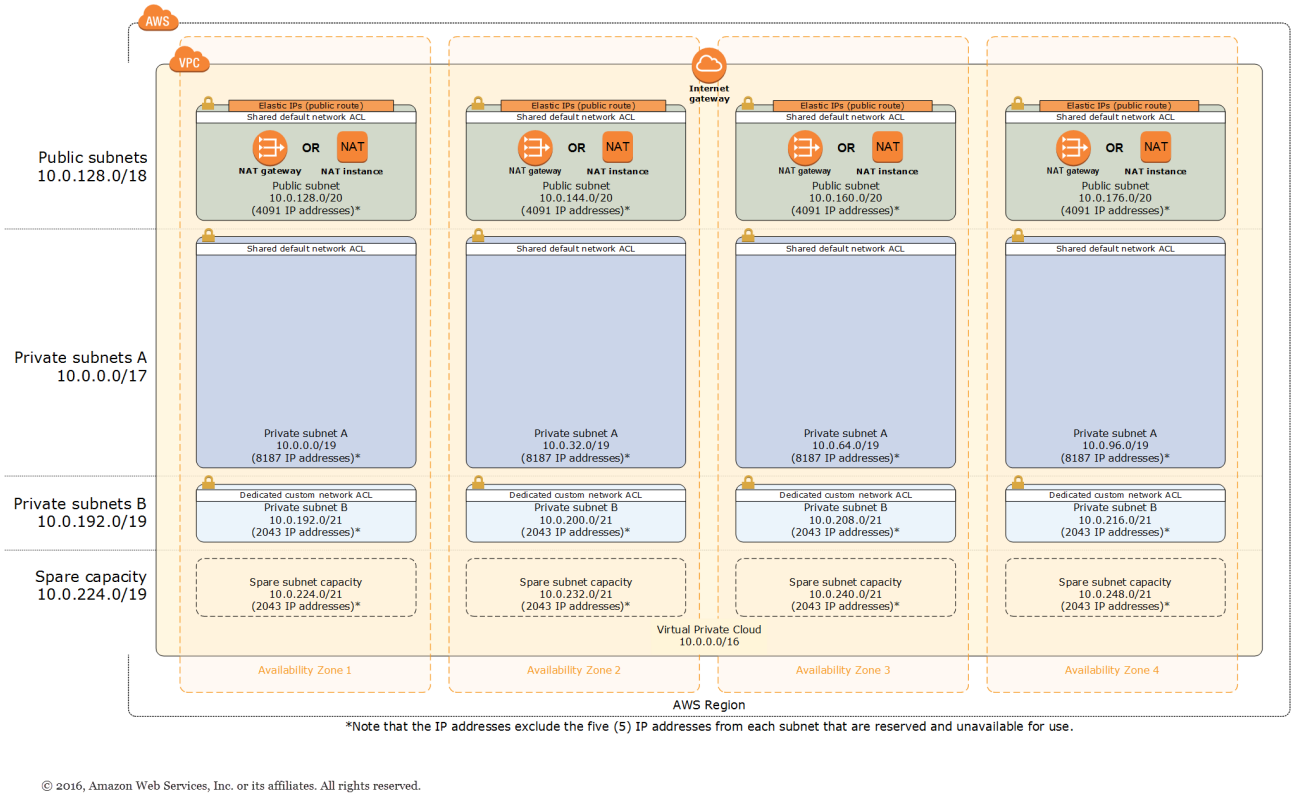


Figura 4-8. Diseño topológico de las subredes definidas por las plantillas de VPC. [43]

4.1.5 Instancias bastión

Puesto que uno de los grupos de requisitos en los que más se insiste es la seguridad de la solución, se propone la separación de redes públicas y privadas, siendo en estas últimas donde se deben desplegar los recursos de forma que estén aislados de internet. Por supuesto, si se quiere ofrecer un servicio web, ciertas partes de la infraestructura deben estar expuestas (por ejemplo, el balanceador de carga).

Para ofrecer un punto de unión entre las redes públicas y las privadas y así poder tener acceso a las instancias, se crean unos recursos denominados *Bastion hosts* que son instancias que sirven como puente para establecer conexiones de acceso entre los administradores de la infraestructura y las máquinas virtuales que ejecutan los servicios (red privada). Al igual que ocurre con las plantillas de *VPC*, las plantillas que incluyen el despliegue de los bastiones son mantenidas directamente por el equipo de *AWS Quick Starts*.

Su topología es mostrada en la *Figura 4-9* [44]. Como se puede observar, las instancias bastión forman parte de un grupo de auto escalado, de forma que se garantice la existencia de al menos una instancia en alguna de las zonas de disponibilidad. En el caso de los bastiones, al no ser una pieza crítica de cara al usuario final, no se despliega una instancia por zona de disponibilidad, sino que, en caso de fallo, se lanzará una nueva en la zona de disponibilidad que sea funcional.

Si se observan las instancias desplegadas por la solución *WordPress High Availability* (*Figura 4-10*), se puede ver como solamente las instancias bastión poseen una dirección *IP* pública.

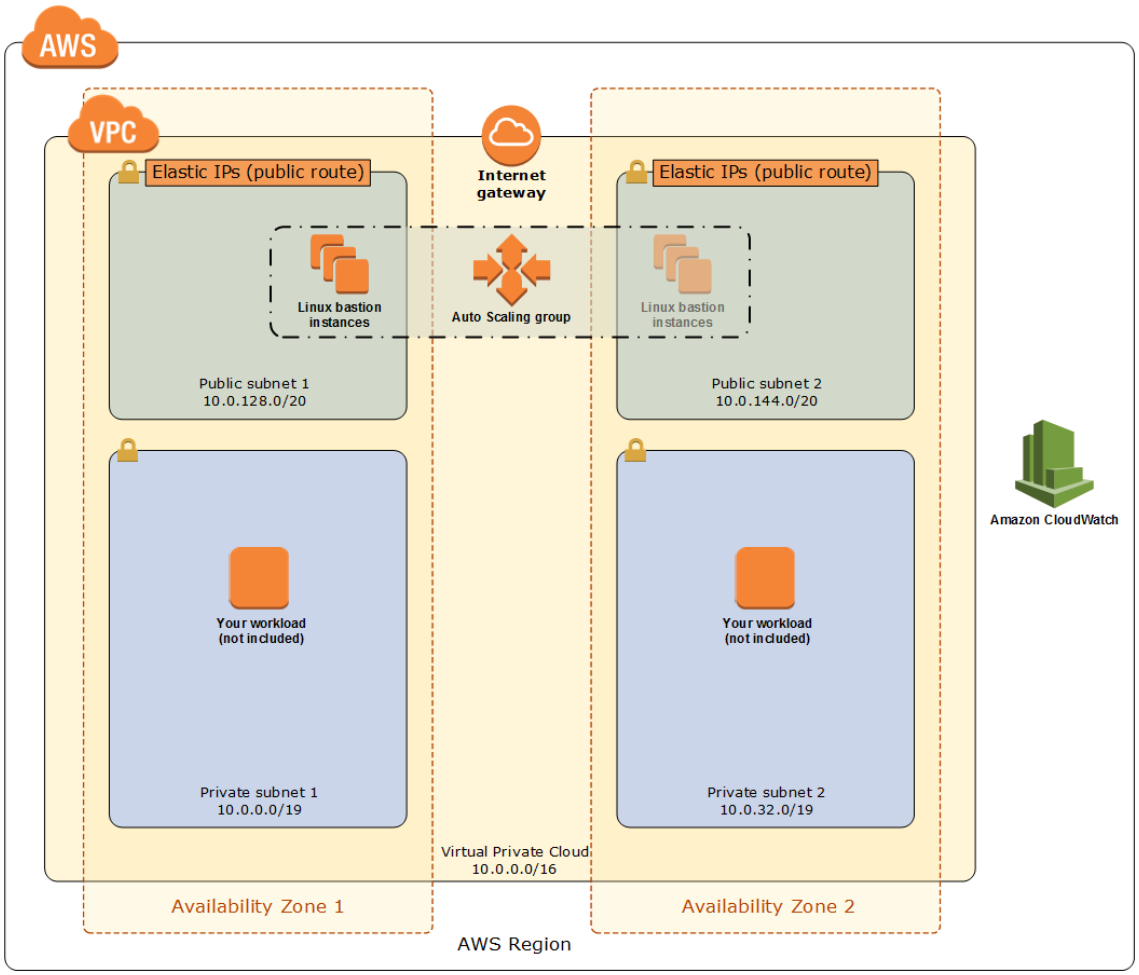


Figura 4-9. Diagrama de arquitectura de los equipos bastion. [44]

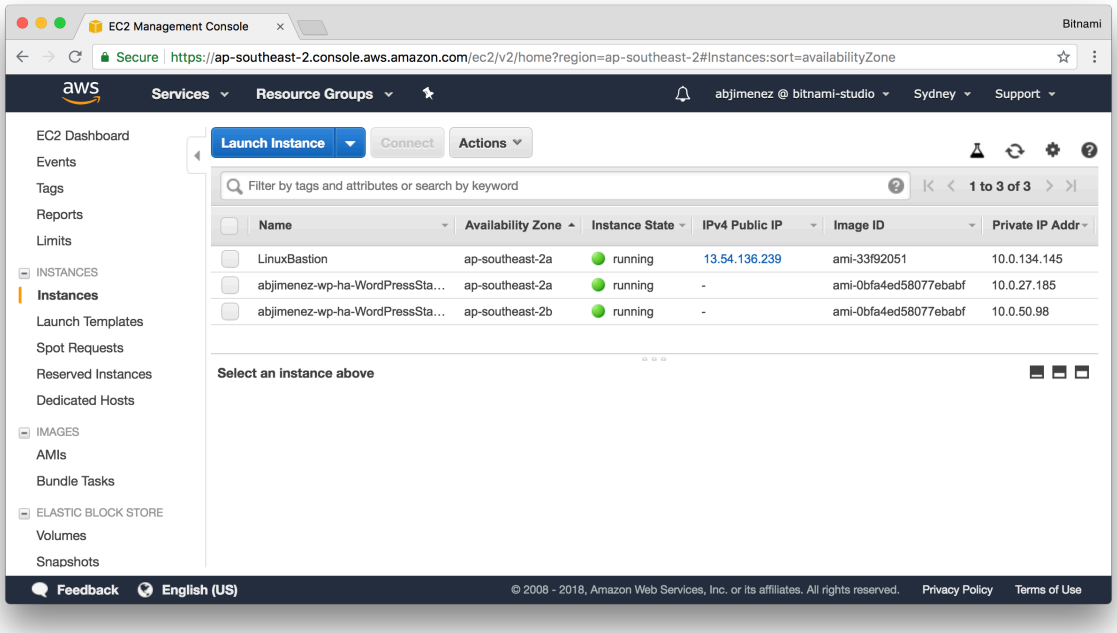


Figura 4-10. Vista de la consola de AWS mostrando instancias bastión.

4.1.6 Balanceo de carga

Hasta este punto, se ha descrito una arquitectura con múltiples servidores web en una red privada. Los usuarios finales no tendrían modo de acceder a la web desplegada. Debe existir un punto de acceso público y que este sea preferiblemente único, un único nombre de dominio (*Figura 4-11*).

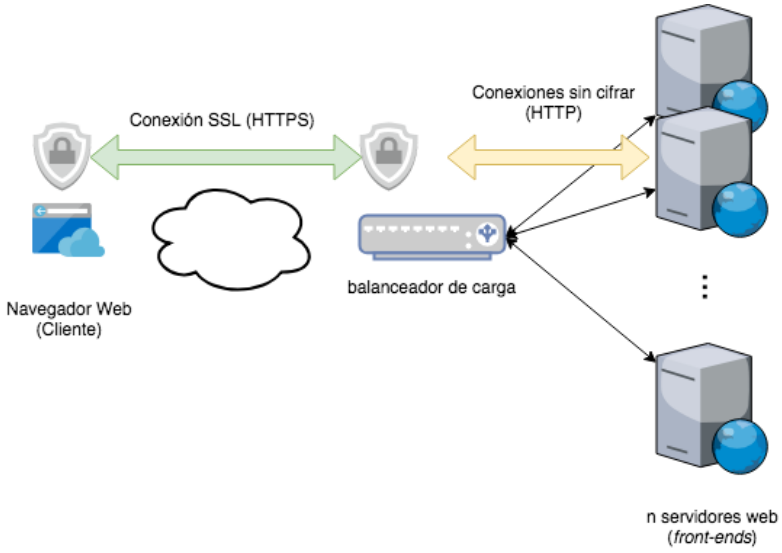


Figura 4-11. Balanceador de carga como acceso al servicio.

Por el hecho de tener múltiples instancias ejecutando un mismo servicio web (instancias *front-end*), se hace necesario el uso de un balanceador de carga. Además de ser una necesidad,

también sirve para ofrecer un mejor servicio ya que se mejora rendimiento de la aplicación por balancear la carga (como su propio nombre indica) entre varias instancias.

Como se detalla en la sección 3.2.3 sobre la configuración del balanceador de carga de *AWS*, es necesario especificar una serie de grupos de destino y también uno o más *listeners* en los apartados siguientes se analizan las configuraciones de mayor relevancia utilizados en *WordPress High Availability*.

4.1.6.1 Grupos de destino

El grupo de destino se configura de forma que los chequeos de estado se realicen al puerto 80 (*HTTP*) cada 30 segundos (*Tabla 4-4*). Por otra parte, se habilita la característica de "*sticky sessions*" para que las peticiones de una misma conexión se realicen siempre a la misma máquina. Como se ha comentado previamente, este mecanismo hace uso de una *cookie* de sesión. Se puede chequear utilizando *Chrome Development Tools*[35] (*Figura 4-12*).

Tabla 4-4. Configuración del grupo de destino del balanceador de carga.

```
"ALBTargetGroup": {
    "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
    "Properties": {
        "HealthCheckIntervalSeconds": 30,
        "HealthCheckTimeoutSeconds": 5,
        "HealthyThresholdCount": 2,
        "HealthCheckPort": 80,
        "HealthCheckProtocol": "HTTP",
        "Port": 80,
        "Protocol": "HTTP",
        "UnhealthyThresholdCount": 5,
        "VpcId": {
            "Ref": "VPCID"
        },
        "TargetGroupAttributes": [
            {
                "Key": "stickiness.enabled",
                "Value": "true"
            },
            {
                "Key": "stickiness.type",
                "Value": "lb_cookie"
            },
            {
                "Key": "stickiness.lb_cookie.duration_seconds",
                "Value": "30"
            }
        ]
    }
},
```

[35] Conjunto de herramientas que forman parte del navegador Chrome para el análisis de sitios web y diagnóstico de errores.

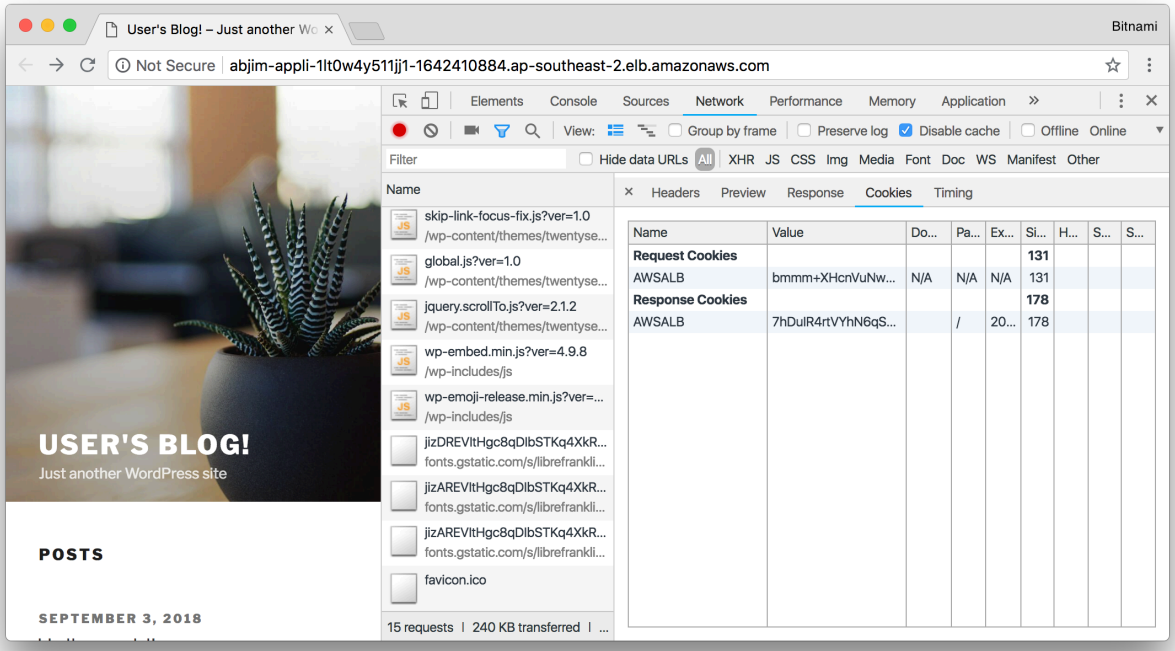


Figura 4-12. Uso de la cookie AWSALB para el mantenimiento de conexiones con una misma instancia.

4.1.6.2 Listeners

En *WordPress High Availability* se crea un único *listerner*. Se configura que la acción a realizar sea una simple redirección al grupo de destino. Por otra parte, también se configura el puerto de escucha del balanceador de carga. El puerto se configura de forma condicional. Será el puerto 80 (*HTTP*) o el puerto 443 (*HTTPS*) según lo que indique la condición *UseSSL* (condición basada en parámetros definidos por el usuario). En el caso de usar *SSL*, se indica el certificado *SSL* correspondiente. El extracto de esta configuración se muestra en la *Tabla 4-5*.

Tabla 4-5. Configuración del listener del balanceador de carga.

```
        "ALBListener": {
            "Type": "AWS::ElasticLoadBalancingV2::Listener",
            "Properties": {
                "DefaultActions": [
                    {
                        "Type": "forward",
                        "TargetGroupArn": {
                            "Ref": "ALBTargetGroup"
                        }
                    }
                ],
                "LoadBalancerArn": {
                    "Ref": "ApplicationLoadBalancer"
                },
                "Port": { "Fn::If": [ "UseSSL", 443, 80 ] },
                "Protocol": { "Fn::If": [ "UseSSL", "HTTPS", "HTTP" ] },
```

```
                "Certificates": [
                    { "Fn::If": [ "UseSSL",
                        {
                            "CertificateArn": {
                                "Fn::If": [ "GenerateSSLCertificate",
                                    { "Ref": "ACMCertificate" },
                                    { "Ref": "SSLCertificateId" }
                                ]
                            }
                        },
                        { "Ref": "AWS::NoValue" }
                    ] }
                ]
            }
        },
```

Como se puede ver en las configuraciones de los grupos de destino y en los *listeners*, las conexiones entre el balanceador de carga y las instancias utilizan el protocolo *HTTP*, independientemente del protocolo utilizado entre el usuario final y el balanceador (*Figura 4-11*).

La razón de esto es que el manejo del cifrado y descifrado de las comunicaciones *SSL* son muy costosas computacionalmente hablando ya que hacen un uso intensivo de la *CPU*. Por ello, es conveniente delegar esa tarea de cifrado a los nodos balanceadores de carga y dedicar las instancias internas a servir el sitio web.

Gracias al uso de redes privadas dentro de una *VPC*, se puede considerar que es aceptable el uso de un protocolo sin cifrado para conexiones internas.

4.2 Diagrama de arquitectura

Presentadas las soluciones aportadas para poder ofrecer una aplicación tradicional con todas las ventajas de la infraestructura cloud, en la *Figura 4-13* se muestra la arquitectura completa de *WordPress High Availability*:

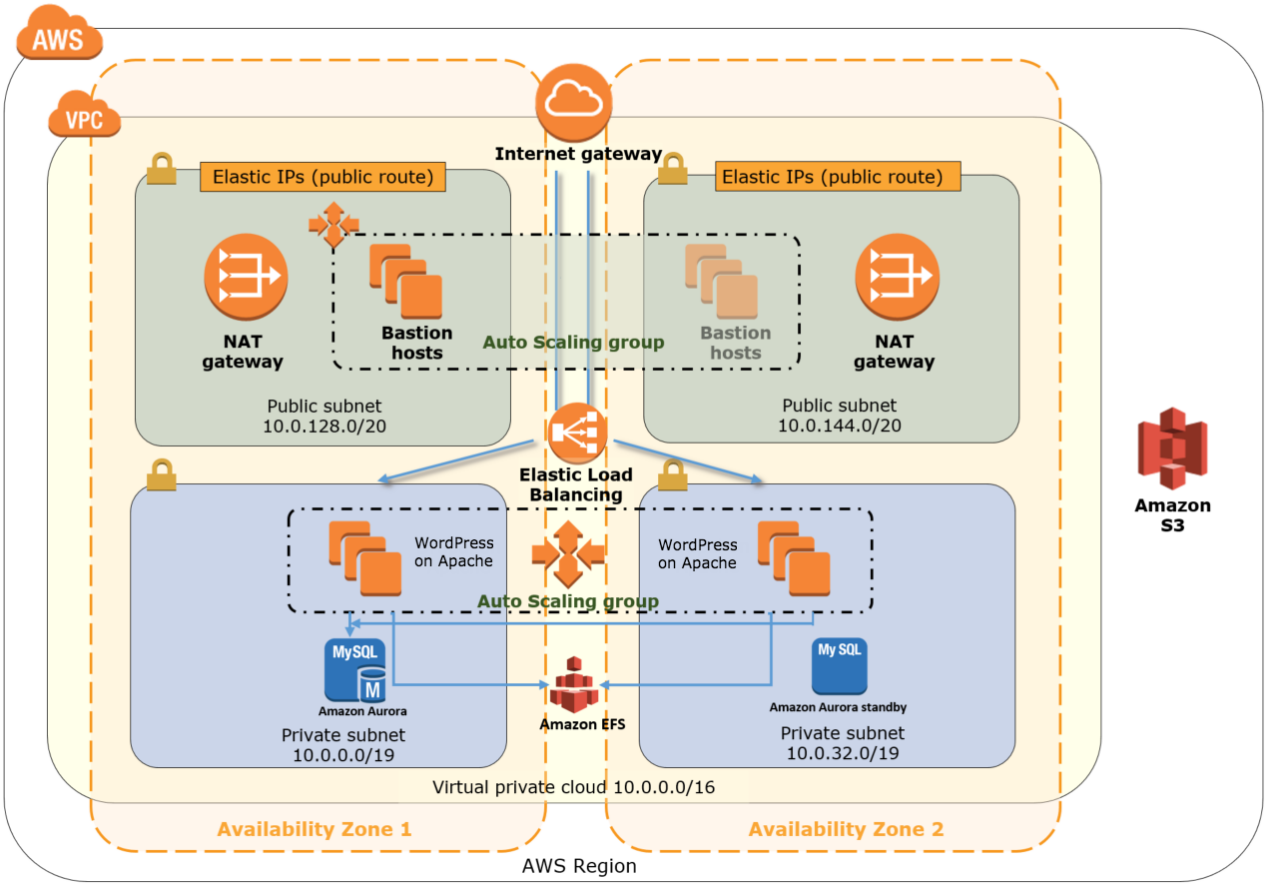


Figura 4-13. Diagrama de arquitectura de la solución WordPress High Availability.

En el diagrama se pueden apreciar diferentes secciones. A nivel general, se presenta la región de *AWS* donde se despliega la solución. Esta se subdivide en dos zonas de disponibilidad (recuadros naranjas en línea discontinua). Cada una de estas zonas de disponibilidad contiene una subred pública (recuadro verde) y otra privada (recuadro azul).

Además de estas subdivisiones, se representan los diferentes servicios utilizados y que han sido descritos previamente. Cabe destacar la presencia de los elementos "*WordPress on Apache*" que forman parte de un grupo de escalado automático. Se trata de las instancias que ejecutan el servicio web. Serán descritas en más detalle posteriormente.

5 FUNCIONALIDADES PRINCIPALES Y VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se pretende enumerar y dar a conocer las funcionalidades principales que tiene la solución *WordPress High Availability*, mostrar como funcionan estas características y describir como se comporta la solución ante determinadas situaciones como, por ejemplo, alta demanda del servicio.

Para poner en valor sus múltiples ventajas, en la *Tabla 5-1* se realiza una comparativa entre *WordPress Stack*, un producto que Bitnami ya ofrece en la *cloud AWS,* y la nueva solución *WordPress High Availability*.

La solución ya existente, *WordPress Stack*, consiste en una única instancia de *cloud* que contiene preinstalada la aplicación *WordPress* junto con la base de datos *MySQL*. Ambos servicios están listos para ser usados por los usuarios ya que su configuración se realiza de forma automática. Su precio es mucho más reducido que el de la nueva solución, pero también son más reducidas sus funcionalidades como se muestra en la siguiente tabla.

Tabla 5-1. Comparativa de características entre la stack de WordPress tradicional y la nueva solución WordPress High Availability.

Característica	WordPress Stack	WordPress High Availability
Despliegue sencillo	✓	✓
Tiempo de despliegue inicial	✓ ~5 minutos	✗ ~20 minutos
Gestión automática de DNS	✗	✓
Actualizaciones automáticas	✗	✓
Alta disponibilidad	✗	✓
Generación y gestión de certificados SSL	✗	✓
Copias de respaldo de la base de datos	✗	✓
Precio	✓ ~30 $/mes	✗ 100-150 $/mes

En las siguientes secciones se describen las funcionalidades principales de la nueva solución.

5.1 Despliegue sencillo de WordPress

La primera funcionalidad que merece mención, aunque obvia, es la de desplegar un sitio web basado en la aplicación *WordPress* de manera muy sencilla. Existen diversos métodos con los que desplegar la solución. En cualquiera de ellos, lo primero que se debe hacer es obtener las plantillas *Quick Start* de la solución. Esto se puede hacer a través del sitio https://aws.amazon.com/quickstart/ o directamente a través del repositorio de código de *Quick Start* alojado en *GitHub*: https://github.com/aws-quickstart/quickstart-bitnami-wordpress[36] (*Figura 5-1*).

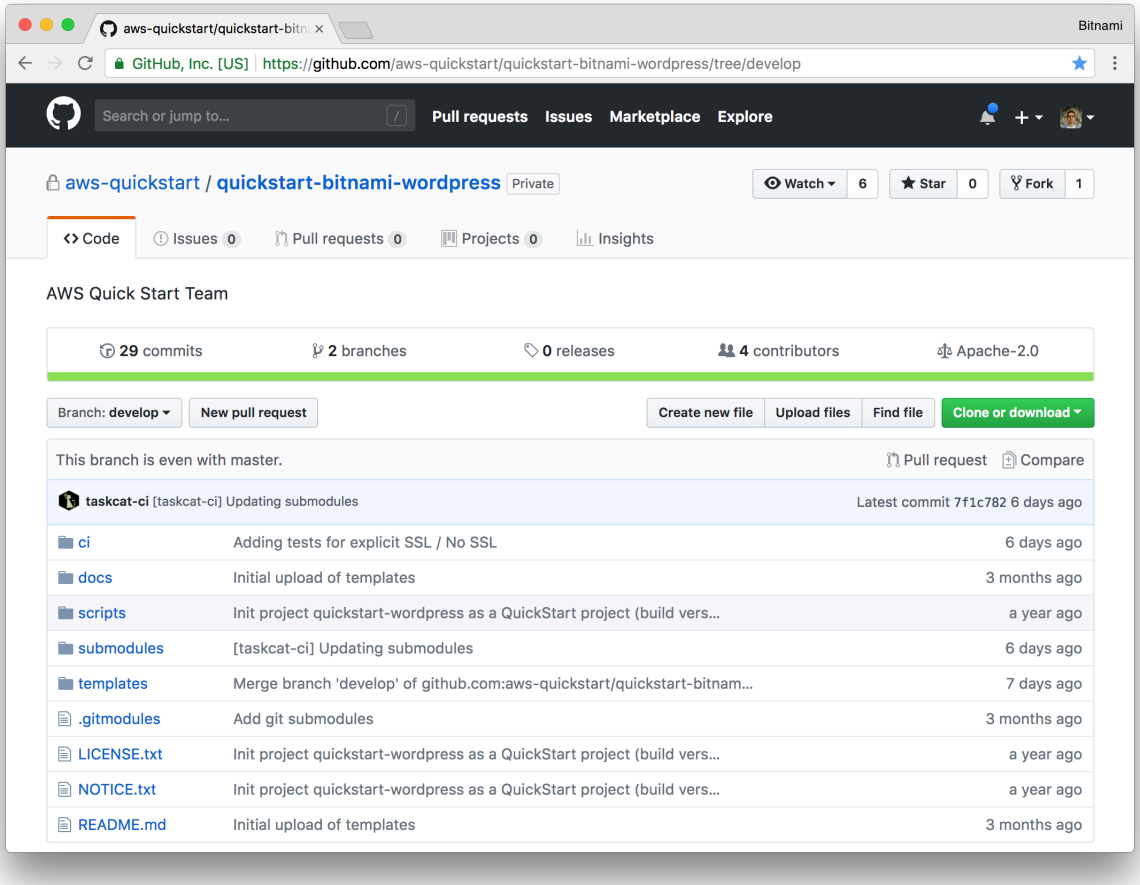


Figura 5-1. Repositorio de código que contiene las plantillas de la solución WordPress High Availability.

5.1.1 Despliegue utilizando el portal de AWS

Tras crear una cuenta de *AWS*, se puede acceder a la consola de *AWS*, en concreto, al servicio *CloudFormation* accesible a través de https://console.aws.amazon.com/cloudformation/, donde se puede indicar la plantilla principal de la solución que, a su vez, referencia a otras plantillas.

[36] El repositorio, aunque existe, es aún privado puesto que la solución todavía se encuentra en fase de revision.

Una vez que la plantilla es procesada por el servicio, dependiendo de los parámetros definidos en ella, se muestran una serie de parámetros que el administrador del despliegue debe proporcionar (*Figura 5-2*). Algunos de los parámetros más interesantes son los que permiten generar un certificado *SSL* de forma automática o los parámetros relativos al grupo de escalado automático, que permiten establecer un número máximo y mínimo de instancias que tener en ejecución.

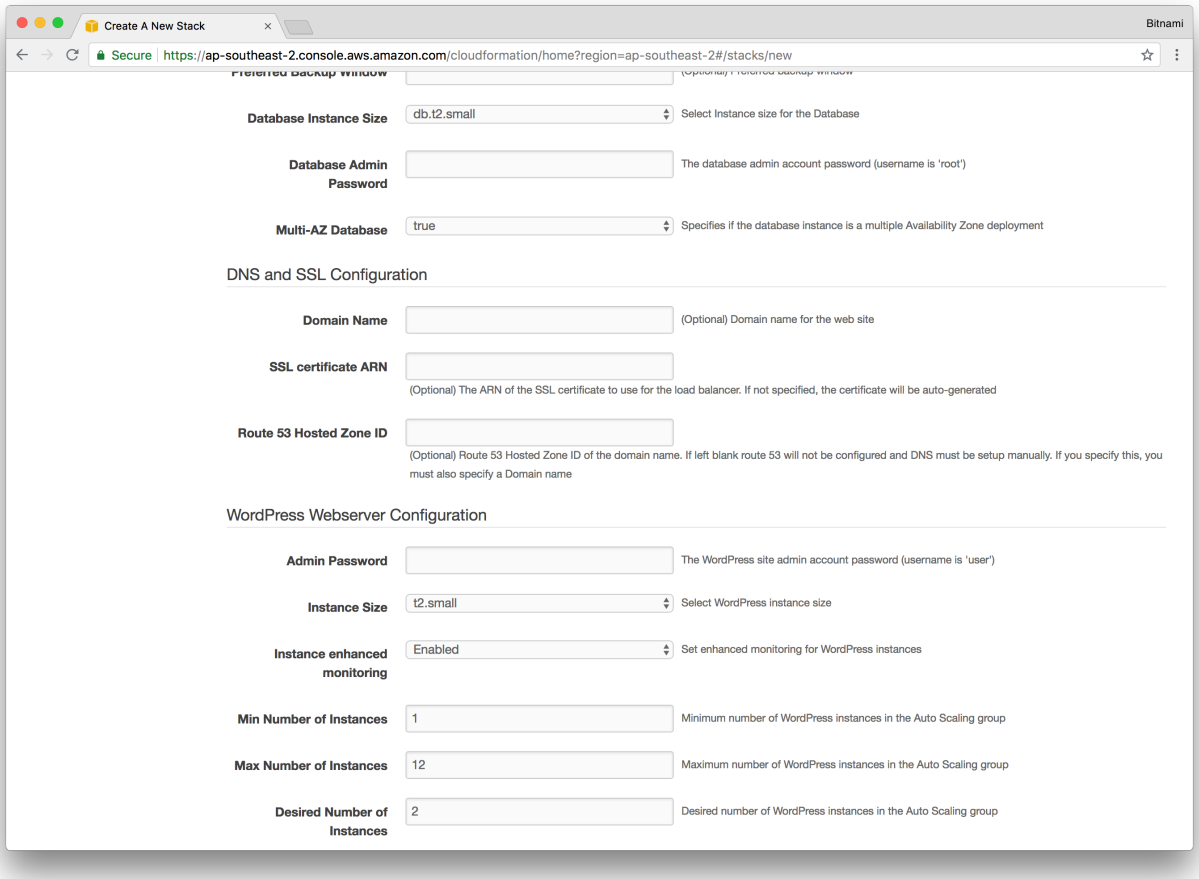


Figura 5-2. Vista de parámetros a introducir para personalizar el despliegue.

5.1.2 Despliegue utilizando aws-cli

Otro método para realizar un despliegue con *CloudFormation* es utilizar la herramienta *aws-cli*. Para ello, primero se debe clonar[37] el repositorio de Quick Start *WordPress* para posteriormente ejecutar la utilidad utilizando los ficheros clonados (*Figura 5-3*). El comando utilizado es *cloudformation create-stack* y la información que se provee a través del portal de *AWS* en el método anterior, debe ser dada por medio de un fichero (*parameters.json*).

[37] En el uso de repositorios Git, clonar significa crear una copia local de un repositorio remoto.

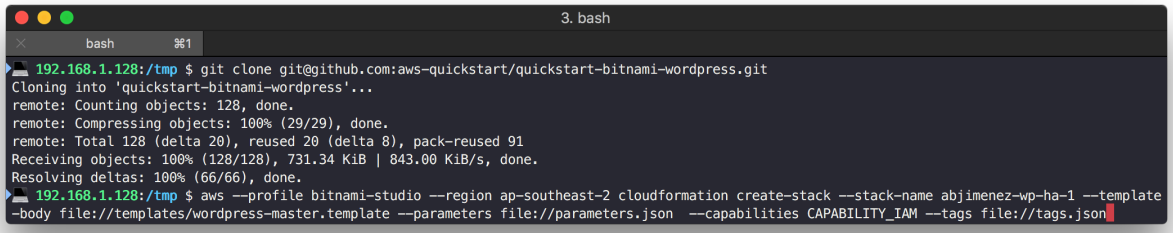


Figura 5-3. Despliegue de la solución utilizando aws-cli.

5.2 Actualización de la solución

CloudFormation permite realizar una actualización de los recursos ya en funcionamiento. Para ello, simplemente es necesario volver a desplegar la nueva versión de las plantillas. De nuevo, esta acción se puede realizar mediante el portal o con la herramienta *aws-cli*. En el proceso de actualización, se analizan todos los recursos y si alguno es necesario cambiarlo, se destruye y se crea uno nuevo.

El caso más común para el que se puede requerir una actualización es la aparición de una nueva versión de la aplicación (*Figura 5-4*). En cuanto las nueva *AMI* de la versión más reciente se hace pública, esta se puede actualizar. Puesto que el cambio solo se produce en las instancias, el único recurso implicado en el proceso de actualización es el grupo de escalado automático, quedando el resto de recursos intacto.

La solución *WordPress High Availability* permite realizar esta actualización siguiendo un orden concreto y de forma automática. Esta técnica de actualización se denomina *Rolling Update*. Permite mantener el servicio en todo momento puesto que las instancias más antiguas se van eliminando de forma intercalada con las nuevas que se crean (*Figura 5-5*). En la *Tabla 5-2* se muestran los distintos estados por los que pasaría un despliegue con 2 instancias *front-end*.

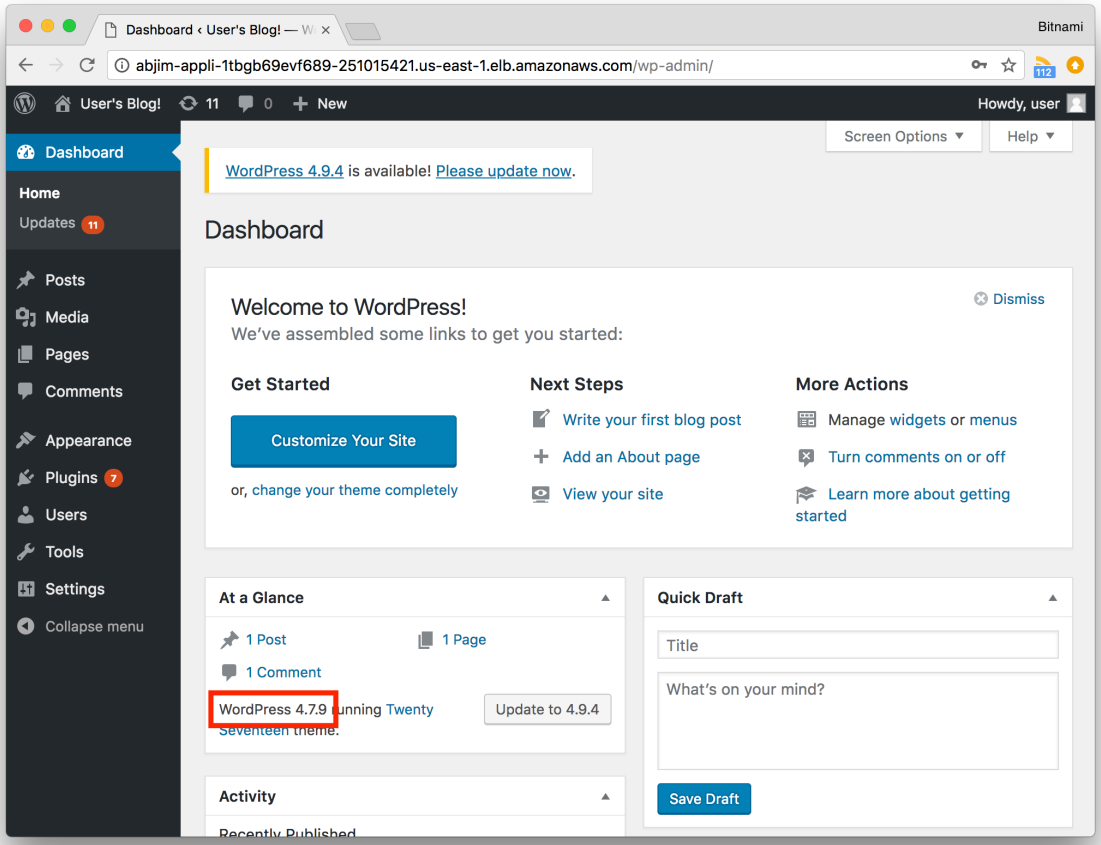


Figura 5-4. Panel de administración de WordPress indicando que hay una nueva versión disponible.

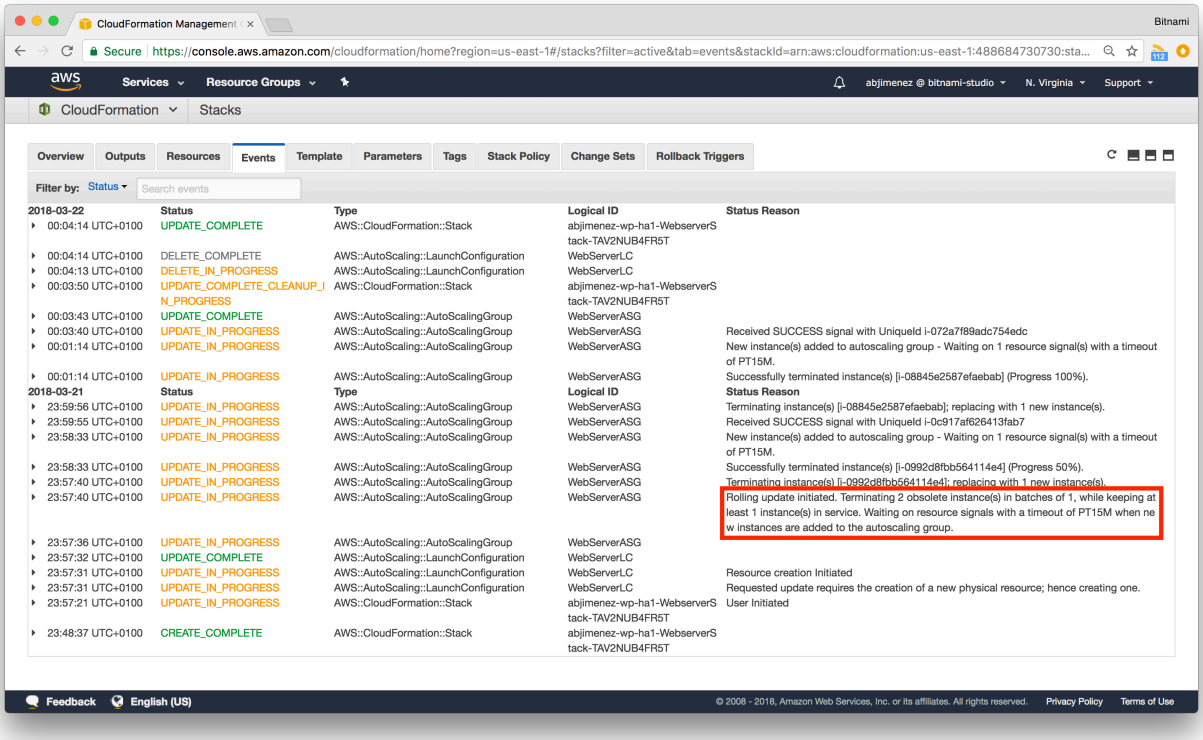


Figura 5-5. Eventos de CloudFormation durante una actualización de instancias mediante Rolling Update.

Tabla 5-2. Estados de las instancias durante el proceso de actualización mediante Rolling Update.

Fases	Instancias	Comentarios
1		2 instancias requieren actualización.
2		1 instancia obsoleta se comienza a eliminar al mismo tiempo que se empieza a crear una de la nueva versión.
3		Se espera hasta que se complete la creación de la nueva instancia.
4		Se repite el paso 2.
5		Se finaliza el proceso habiendo mantenido el servicio en todo momento.

5.3 Copias de respaldo de la base de datos

Esta solución permite crear copias de respaldo de la base de datos alojada en el servicio *RDS*. En caso de haber alguna incidencia en la que la base de datos termine eliminada o en un estado inconsistente, se puede restaurar acudiendo a un estado anterior de esta. La configuración del servicio de estas copias de seguridad se especifica jugando con los parámetros *BackupRetentionPeriod* y *PreferredBackupWindow* del recurso *RDS DBCluster* (*Tabla 5-3*).

Tabla 5-3. Parámetros de configuración de copias de respaldo del clúster RDS.

```
"AuroraDBCluster": {
    "Type": "AWS::RDS::DBCluster",
    "Properties": {
        "BackupRetentionPeriod": {
            "Ref": "DBBackupRetentionPeriod"
        },
        "PreferredBackupWindow": {
            "Ref": "DBPreferredBackupWindow"
        },
```

5.4 Escalado horizontal de instancias.

El concepto de escalado horizontal se refiere a escalar la capacidad de un servicio incrementando el número de recursos disponibles. En esta solución, se refiere al aumento o disminución del número de instancias dentro del grupo de escalado automático. Este escalado se puede hacer de forma manual o automática, basando la decisión en el uso de *CPU* de las instancias (*Tabla 5-4*).

Tabla 5-4. Definición de política de escalado basada en uso de CPU.

```
"WebServerTargetTrackingScalingPolicy": {
    "Type": "AWS::AutoScaling::ScalingPolicy",
    "Properties": {
        "AutoScalingGroupName": {
            "Ref": "WebServerASG"
        },
        "Cooldown": "60",
        "PolicyType": "TargetTrackingScaling",
        "TargetTrackingConfiguration": {
            "PredefinedMetricSpecification": {
                "PredefinedMetricType": "ASGAverageCPUUtilization"
            },
            "TargetValue": 75.0
        }
    }
}
```

Para validar esta caracterísitica, se puede forzar el uso de CPU con la herramienta *stress* que, mediante la realización de operaciones matemáticas (raíces cuadradas) permite simular una situación en la que debido a una alta demanda del servicio se requiere un alto uso de CPU. En la *Figura 5-6* se muestra la ejecución de la herramienta, y en la *Figura 5-8* se puede apreciar como el servicio de monitorización de *AWS, CloudWatch,* detecta un uso de CPU superior al 75% (ver

Tabla 5-4). Ante esta situación, el grupo de escalado automático actúa en consecuencia lanzando una nueva instancia con la que poder balancear el tráfico solicitado *Figura 5-9*.

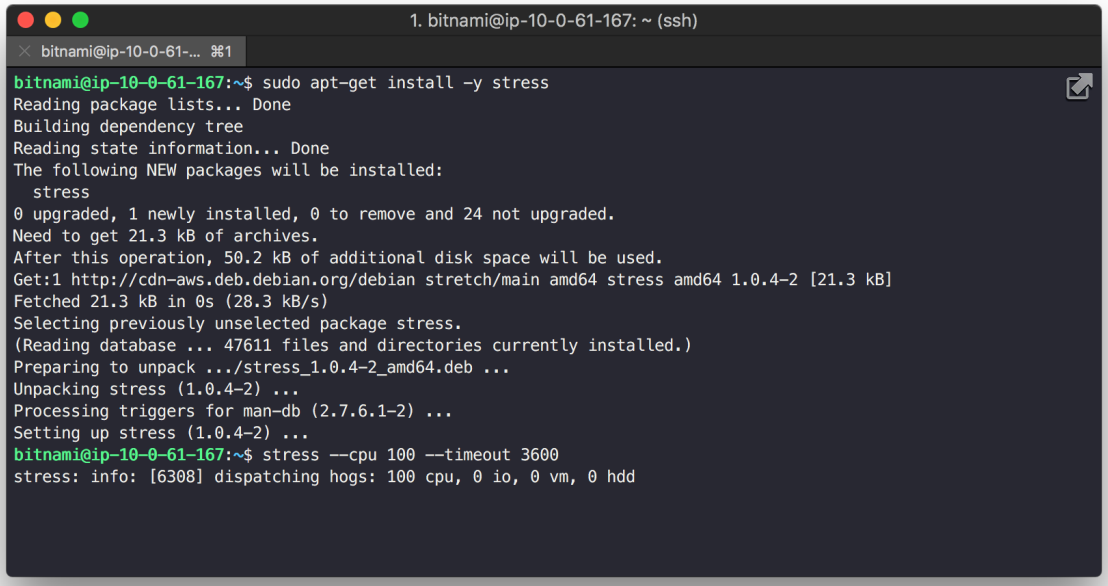


Figura 5-6. Instalación y ejecución de la herramienta stress en las máquinas virtuales.

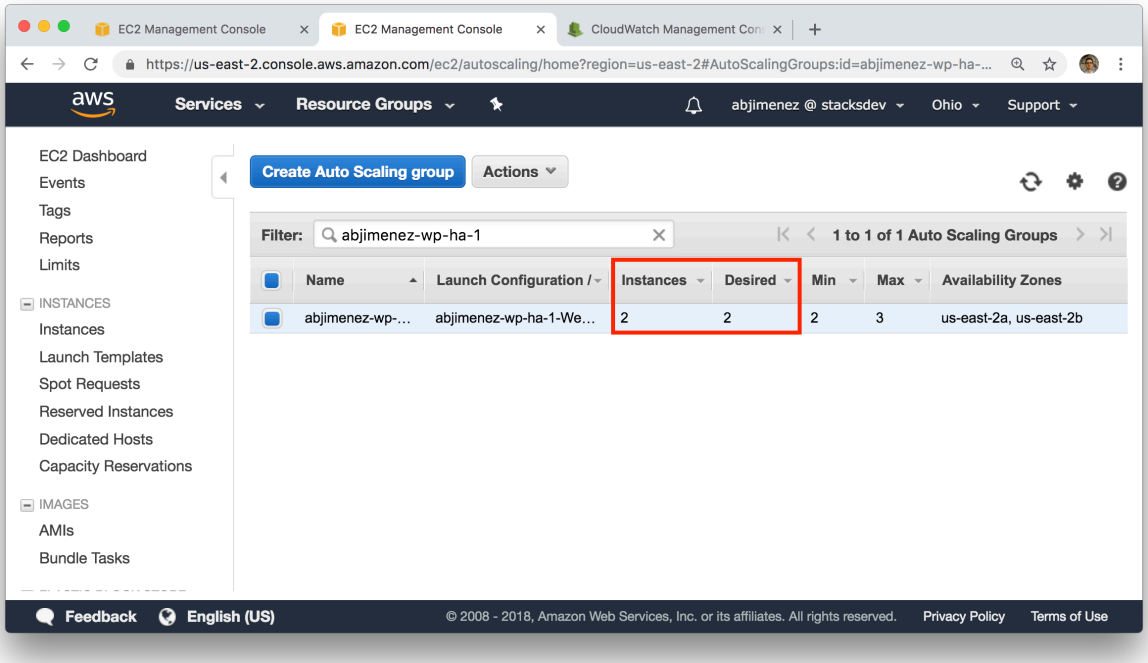


Figura 5-7. Estado inicial del grupo de escalado automático formado por 2 instancias.

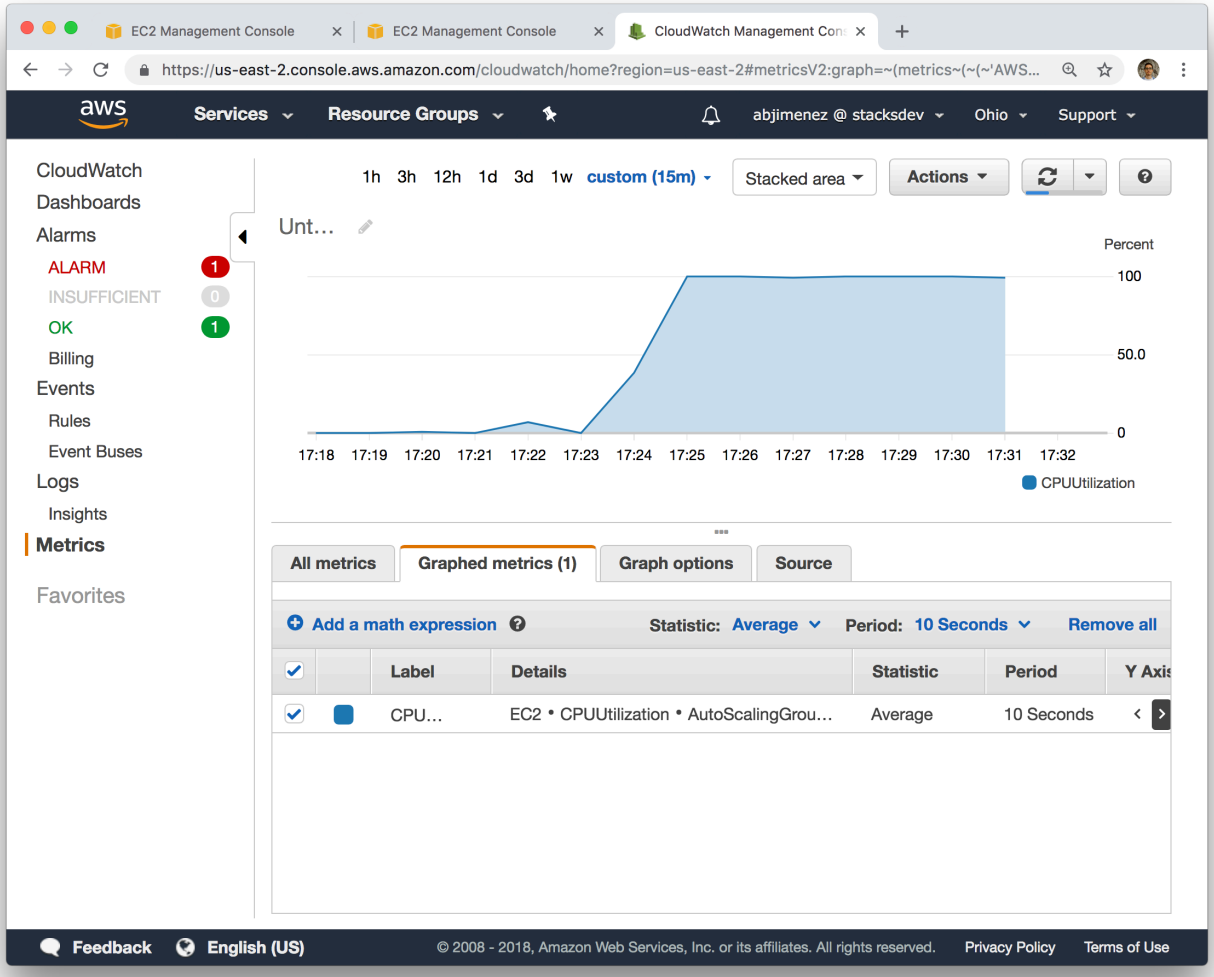


Figura 5-8. Monitorización mediante CloudWatch del porcentaje de uso de CPU en las instancias del grupo de escalado automático.

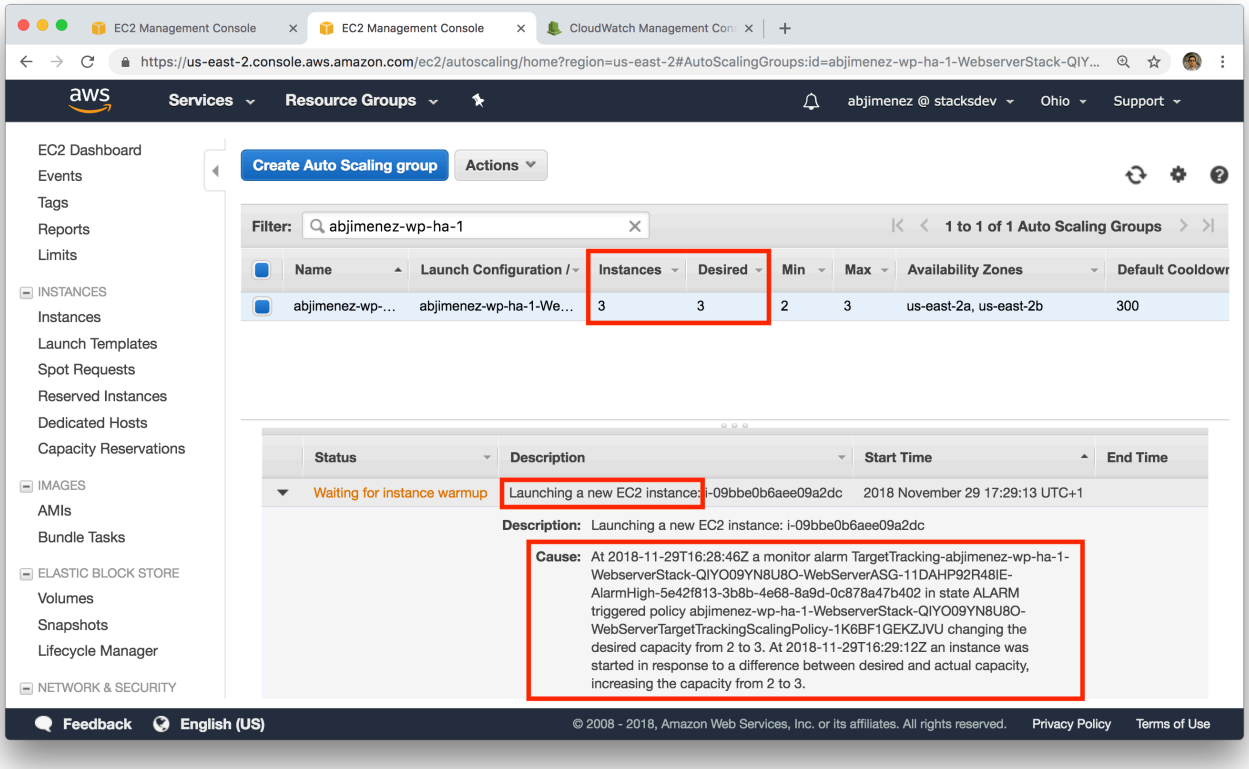


Figura 5-9. Lanzamiento de una nueva instancia en el grupo de escalado automático a causa del aumento del uso de CPU.

5.5 Generación automática de certificados SSL

Si el usuario lo desea, puede generar de forma automática durante la fase de despliegue de la solución un certificado *SSL*. El usuario solo tiene que indicar el dominio que se desea asociar al sitio. Este nombre de dominio puede ser añadido en el servicio de *DNS Route 53* como un alias al dominio del balanceador de carga (*Figura 5-10*).

Para la validación del certificado *SSL* asociado a dicho dominio se envía un correo electrónico a las direcciones de administración asociadas al dominio (*Figura 5-11*) y haciendo clic en el enlace de verificación se genera el certificado que es automáticamente asignado al balanceador de carga (*Figura 5-12*).

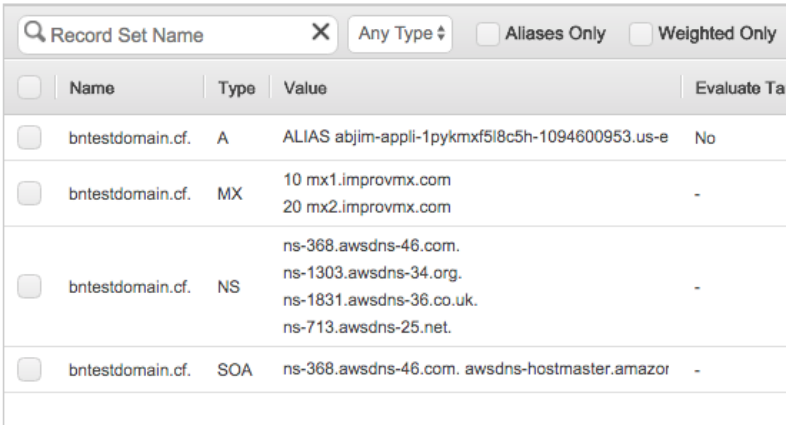
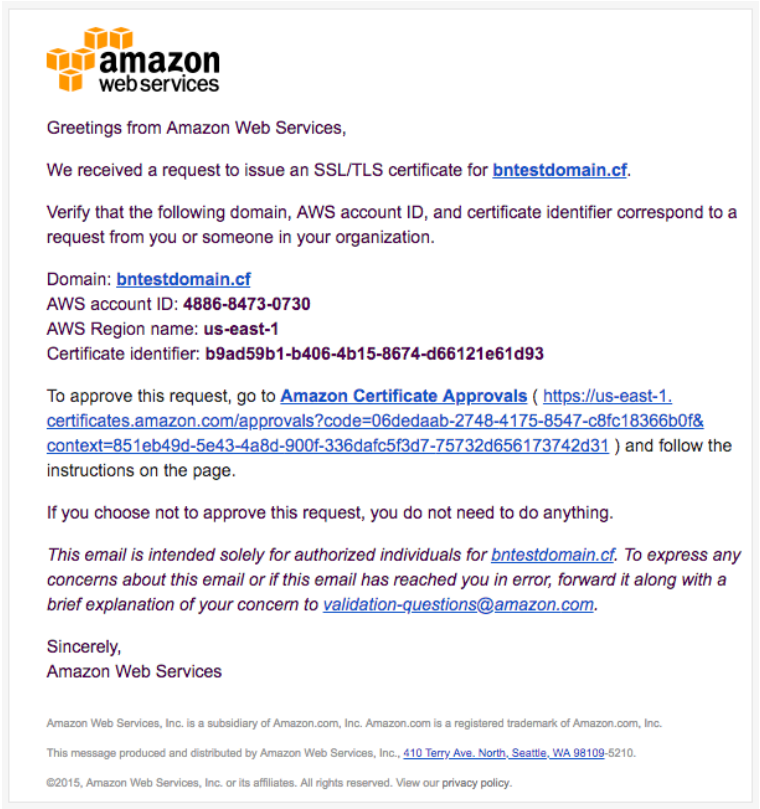


Figura 5-10. Registros DNS de la solución.



amazon web services

Greetings from Amazon Web Services,

We received a request to issue an SSL/TLS certificate for **bntestdomain.cf**.

Verify that the following domain, AWS account ID, and certificate identifier correspond to a request from you or someone in your organization.

Domain: **bntestdomain.cf**
AWS account ID: **4886-8473-0730**
AWS Region name: **us-east-1**
Certificate identifier: **b9ad59b1-b406-4b15-8674-d66121e61d93**

To approve this request, go to **Amazon Certificate Approvals** (https://us-east-1.certificates.amazon.com/approvals?code=06dedaab-2748-4175-8547-c8fc18366b0f&context=851eb49d-5e43-4a8d-900f-336dafc5f3d7-75732d656173742d31) and follow the instructions on the page.

If you choose not to approve this request, you do not need to do anything.

This email is intended solely for authorized individuals for bntestdomain.cf. To express any concerns about this email or if this email has reached you in error, forward it along with a brief explanation of your concern to validation-questions@amazon.com.

Sincerely,
Amazon Web Services

Figura 5-11. Correo electrónico de confirmación para la validación de un certificado SSL.

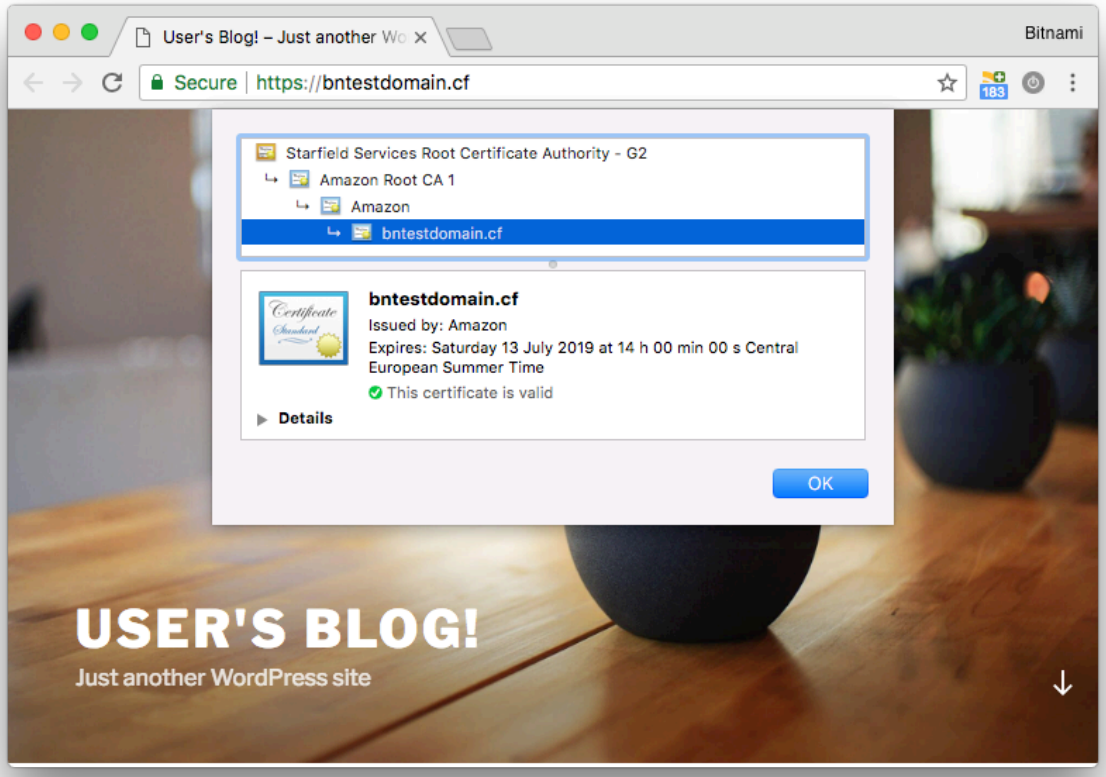


Figura 5-12. Certificado SSL válido generado.

5.6 Alta disponibilidad

Por último, es importante destacar la característica de alta disponibilidad de la solución. Esto se consigue siguiendo la premisa de evitar los puntos únicos de fallo (*SPOF* o *Single Point of Failure* [45]).

Estos se tratan de puntos de un sistema en los que, en situaciones de fallo, harán que todo el sistema completo deje de estar disponible. La solución a esto es añadir redundancia en todos los sentidos. En *WordPress High Availability* algunos puntos del sistema están replicados por defecto gracias al uso de servicios *cloud* (por ejemplo, el balanceador de carga) y otros han sido implementados en las plantillas de *CloudFormation*, por ejemplo, múltiples instancias o múltiples zonas de disponibilidad. Esta multiplicidad implica que, ante un fallo en uno de los sistemas, el sistema completo continúa ofreciendo servicio.

A continuación, se describen dos pruebas realizadas en las que se simula una situación de fallo o alta demanda y se analiza como se comporta la solución ante esta.

5.6.1 Validación 1: Eliminación manual de una instancia dentro del grupo de escalado automático.

Una simple prueba que se puede realizar es la simulación de un fallo en una de las zonas de alta disponibilidad de la *cloud AWS*. Ante esta situación, las instancias alojadas en esa zona quedarían inaccesibles.

La prueba realizada consiste en lo siguiente:

- Acceso a la web desplegada e inicio de una sesión.
- Mediante la monitorización del fichero de logs de acceso de Apache se puede saber qué instancia está sirviendo las peticiones. Debido a la característica de *Sticky Sessions* (ver 3.2.3.2) del balanceador de carga, todas las peticiones se realizan a la misma instancia.
- Se elimina dicha instancia desde el portal de *AWS*.
- Acto seguido, se accede a la web de nuevo. La sesión se mantiene activa y el usuario no aprecia una parada del servicio. Esto es debido a que la sesión se mantiene como una *cookie* en la parte cliente y como una sesión *PHP* en la parte servidora. De no existir la sesión *PHP*, la *cookie* se autentica contra la base de datos que es compartida entre las instancias.
- Como consecuencia de la parada de la instancia, una nueva instancia se comienza a lanzar (ver *Figura 5-13*).

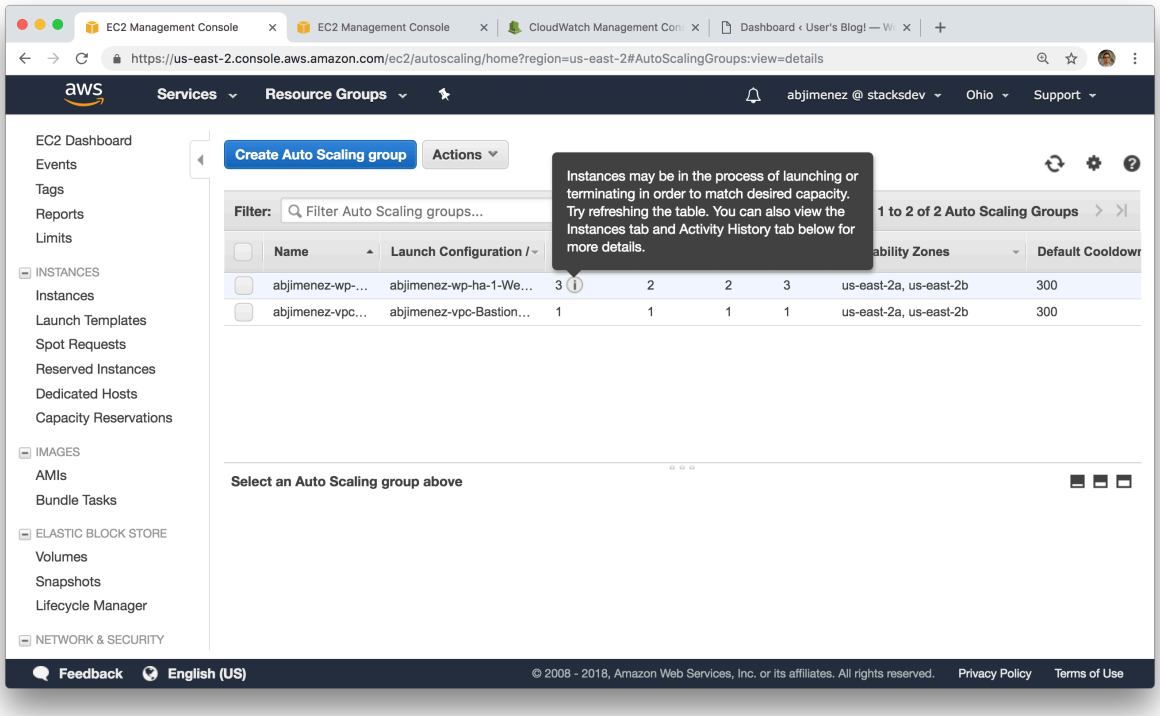


Figura 5-13. Una nueva instancia es lanzada en cuanto se manda la orden de terminación de la instancia.

Mediante esta prueba se demuestra la alta disponibilidad del servicio ante fallos en una de las instancias o zonas de disponibilidad.

5.6.2 Validación 2: Comparativa del tiempo medio de respuesta entre WordPress Stack y WordPress High Availability ante situaciones de alta demanda.

También alta disponibilidad significa que el servicio siga con un funcionamiento adecuado incluso en situaciones de alta demanda. No es aceptable que para la carga de una web se deba esperar más de unos pocos segundos. Para realizar una validación de esta característica, en la *Figura 5-15* se muestra una gráfica comparativa de los tiempos de respuesta entre las soluciones

WordPress Stack y *WordPress High Availability* en cuanto a tiempos de carga de la página principal de la aplicación WordPress frente al uso de CPU.

Desde luego, es lógico que la respuesta de la solución *WordPress High Availability* sea mejor que la de *WordPress Stack* ya que dispone de más recursos (más instancias) entre los que distribuir la carga. Sin embargo, resulta interesante ver cual es la tendencia de ambas soluciones ante situaciones de estrés.

Para realizar el experimento en un entorno controlado, se lanza una nueva instancia adicional desde la que realizar las peticiones. Esta máquina está situada en la misma región que los despliegues de *WordPress* que van a recibir las peticiones. Esto se hace para minimizar los retardos y la variación de retardos que puedan producirse por el encaminamiento de los paquetes desde un ordenador personal en una región lejana.

La herramienta para ejecutar procesos de *CPU* en las instancias es la utilidad *stress*, ya utilizada en la sección *5.4*. Por otra parte, para obtener datos que se asemejen a los que podría obtener un usuario real, se utiliza el navegador *Chrome* que se ejecuta en la instancia adicional mencionada para realizar las peticiones. Para analizar los tiempos de carga se utiliza *Chrome Development Tools*, integrado en el navegador (ver *Figura 5-14*).

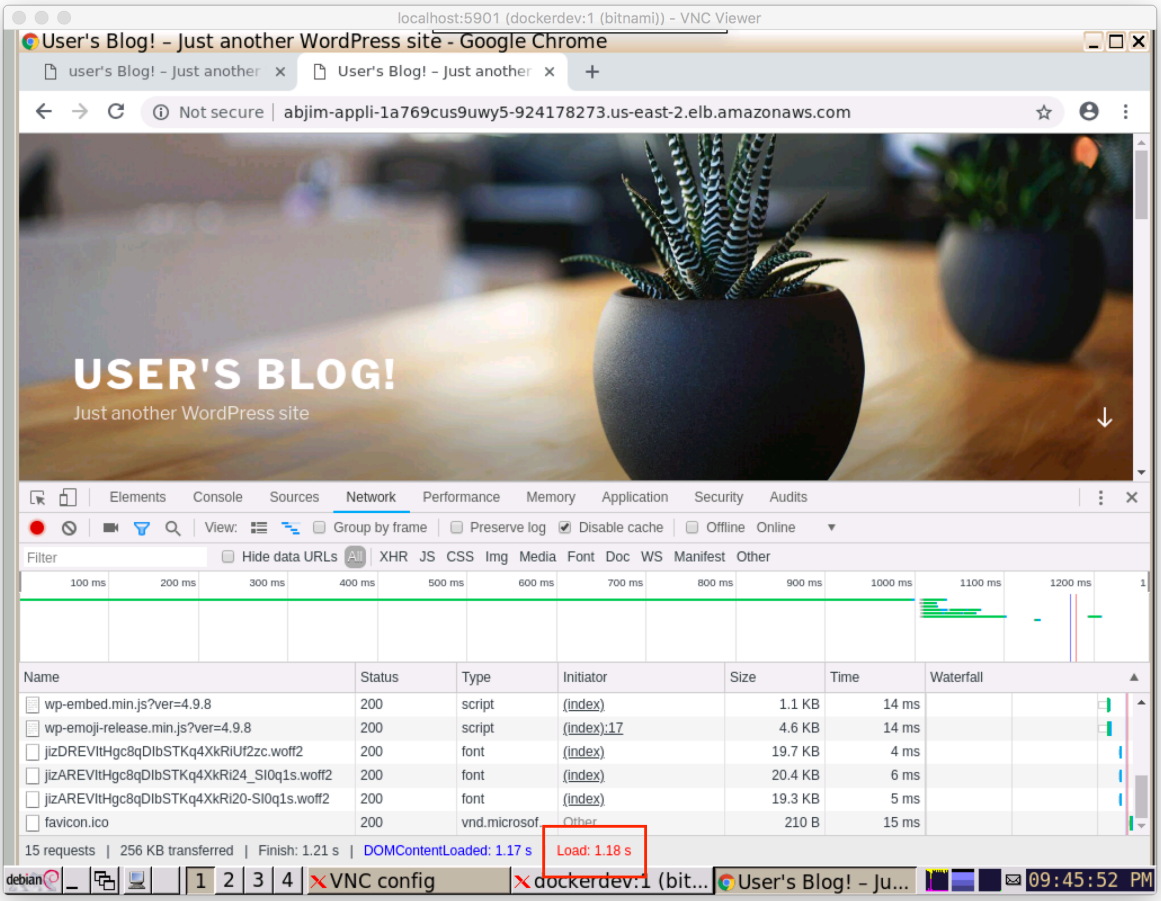


Figura 5-14. Chrome Development Tools utilizado para analizar tiempos de carga.

Para poder controlar el navegador de forma remota en modo visual, se utiliza *VNC* (*Virtual Network Computing*) un programa cliente-servidor de software libre que mediante el protocolo *X11* permite compartir sesiones gráficas.

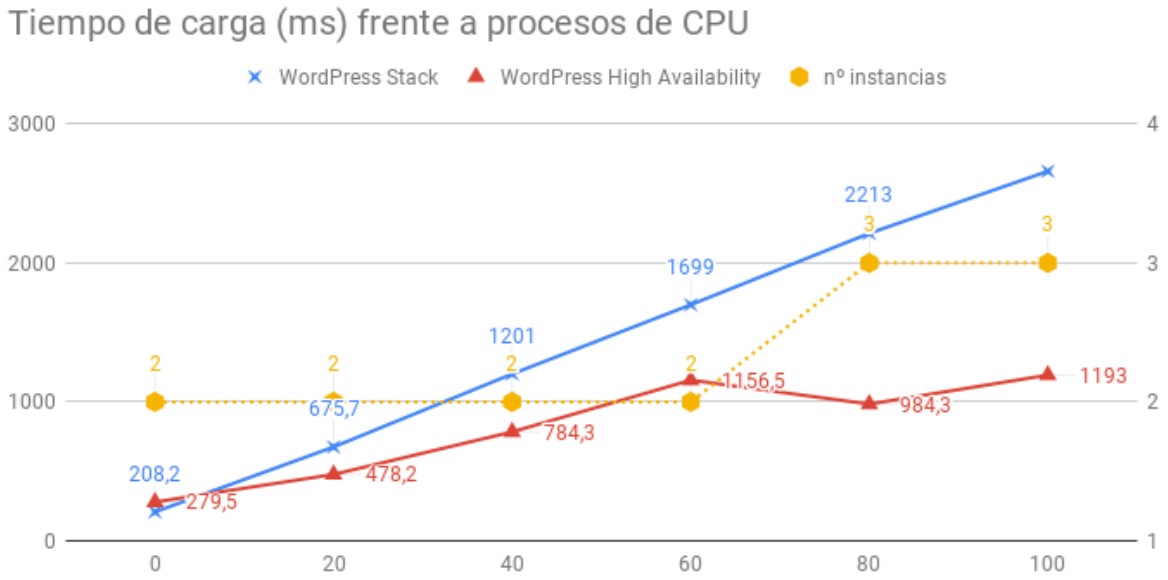


Figura 5-15. Tiempos de carga en milisegundos frente a procesos de CPU en ejecución para las soluciones WordPress Stack y WordPress High Availability.

La gráfica mostrada en la *Figura 5-15* ha sido realizada tomando múltiples muestras (10) para cada punto. Se recogen 3 series que corresponden a los datos para *WordPress Stack* (línea azul con muestras representadas por cruces), *WordPress High Availability* (línea roja con muestras representadas por triángulos) y también se representa en esta gráfica el número de instancias en servicio para *WordPress High Availability* (línea discontinua amarilla con muestras representadas por hexágonos). El eje horizontal representa el número de procesos de *CPU* ejecutados con la herramienta *stress*. El eje vertical izquierdo representa el tiempo de carga en milisegundos. El eje vertical derecho representa el número de instancias.

De los resultados mostrados en la gráfica superior, es interesante destacar algunos efectos relevantes:

- Los tiempos de carga en *WordPress Stack* son inicialmente más reducidos. Este efecto puede deberse a que *WordPress High Availability* posee un balanceador de carga que añade un retardo adicional.
- El tiempo de carga es directamente proporcional a la carga de *CPU* como se observa en la linealidad de la respuesta de *WordPress Stack*.
- Para *WordPress High Availability* se observa un descenso en los tiempos de carga aun cuando el número de procesos de *CPU* aumenta de 60 a 80 procesos. Este descenso es debido a que entra en servicio una nueva instancia entre la que repartir la carga.

Con esta segunda comprobación se puede concluir que efectivamente la solución *WordPress High Availability* responde de una mejor forma ante una situación de estrés. No obstante, es importante destacar que el tráfico recibido por una web para que sea equiparable al uso de *CPU* simulado es realmente alto. Para la mayoría de casos, la solución *WordPress Stack* es más que válida y suficiente.

6 COMPONENTES DE LA IMAGEN (AMI)

Si bien en los capítulos anteriores se describen los servicios *cloud* utilizados en este proyecto, también es importante considerar los componentes instalados en las instancias que ofrecen el servicio web. En este capítulo se pretende describir cada uno de esos componentes.

Además de las aplicaciones instaladas, también es necesario incluir en las instancias una serie de herramientas de configuración que *Bitnami* a desarrollado. Por último, en este capítulo se describe la comunicación que tiene lugar entre las instancias desplegadas y el servicio *CloudFormation*.

6.1 Creación de la imagen

En secciones anteriores (4.1.2) se ha discutido la importancia que tiene la definición y uso de imágenes para esta solución ya que el grupo de escalado automático las utiliza para crear instancias basadas en esas imágenes y poder realizar un escalado horizontal.

En *AWS*, la creación de imágenes [46] se realiza a partir de una instancia ya existente o a partir de un *snapshot* del disco *EBS* raíz. Para ello, se puede utilizar el portal web de *AWS* o la herramienta *aws-cli*. En *Bitnami*, la creación de imágenes la realiza un sistema denominado *BRadmin* de forma automática. Este sistema gestiona todas las integraciones con las diferentes clouds a las que da soporte. Es un proyecto escrito en *Ruby*, por lo que para la creación de imágenes se utiliza el *SDK* de ese lenguaje.

De forma muy resumida, el proceso de creación de la imagen es el siguiente:

- Lanzamiento de una instancia basada en una imagen base. En el caso de *WordPress High Availability*, la imagen base es *Debian*[38]. Sobre esa instancia, por conexiones remotas *SSH* se ejecutan los comandos apropiados para:
    - Creación de nuevos usuarios del sistema.

[38] Debian es un sistema operativo basado en Linux con gran foco en la estabilidad del sistema. Sirve como base para otras distribuciones más conocidas como por ejemplo Ubuntu.

  - Si es conveniente, se realizan modificaciones de parámetros del *kernel*[39] de *Linux* para optimizar el rendimiento.
  - Instalación de los paquetes del sistema que se crean convenientes.
  - Descarga y descompresión de un paquete que contiene los servicios y aplicaciones principales de la imagen. Este paquete ha sido previamente compilado. En el caso de *WordPress High Availability* este paquete va a contener los ficheros de *Apache*, *PHP* y *WordPress* entre otros.
  - Instalación de un agente de estadísticas. Se trata de un binario que envía información no confidencial de la solución lanzada a un servidor centralizado. Esta información sirve de ayuda a *Bitnami* para conocer qué soluciones están funcionando mejor y cuales no en términos de numero de "horas de *cloud*"[40].
  - Se realiza una limpieza de la imagen. Entre las tareas más importantes, y por razones de seguridad para los usuarios finales, se elimina la clave *SSH* utilizada durante el proceso de construcción de la imagen.
- Creación de un *snapshot* del volumen raíz *EBS*.
- Registro del *snapshot* como imagen, inicialmente se registra como una imagen privada, es decir, solo es accesible desde la cuenta de *AWS* de *Bitnami*.

6.2 Configuración automática: Nami

Para ofrecer un sitio web *WordPress* son necesarios varios paquetes de software. Estos son *Apache*, *PHP*, *WordPress* en sí y un cliente *MySQL*. Cada uno de estos 4 paquetes son proyectos de software libre que el usuario puede descargar. Algunos de ellos requieren de preparación previa para convertirse en programas ejecutables. Otros, requieren ser configurados para ser usados.

Para ilustrar esta idea, se puede poner precisamente el ejemplo de *WordPress*. Suponiendo que los prerrequisitos ya se cumplen (*Apache*, *PHP* y *MySQL* funcionando), los pasos a seguir para hacer funcionar *WordPress* según la documentación oficial son los siguientes [47]:

1. Descargar y descomprimir el paquete *WordPress*.
2. Crear una base de datos y un usuario específicos para la aplicación.
3. Editar el fichero de configuración *wp-config.php* con los datos de acceso a la base de datos.
4. Configurar *Apache* para que conozca la localización de los ficheros de *WordPress*.
5. Ejecutar un *script* de instalación guiada a través del navegador.

Aunque, si se tiene cierta experiencia, estos pasos no son complejos de realizar manualmente, en una solución en la que el servicio debe estar disponible con un clic, todos estos pasos deben ser realizados de forma automática. Con este propósito se diseñó una herramienta denominada *Nami* (https://github.com/bitnami/nami).

La herramienta es distribuida bajo una licencia de software libre y está escrita en el lenguaje *JavaScript* (*Node.Js*). *Nami* ofrece una sencilla *API* para configuración de aplicaciones. Esto quiere decir que proporciona una serie de comandos que representan operaciones que realizar

[39] El kernel de un sistema operativo es la parte que tiene control sobre el resto de elementos del sistema. Es la capa más cercana al hardware de la máquina como la CPU, el almacenamiento o la memoria.
[40] Las horas de cloud son una unidad de medida del uso que tienen las soluciones.

sobre las aplicaciones. Algunas de las más importantes son *unpack*, *initialize,* o *start* (*Figura 6-1*).

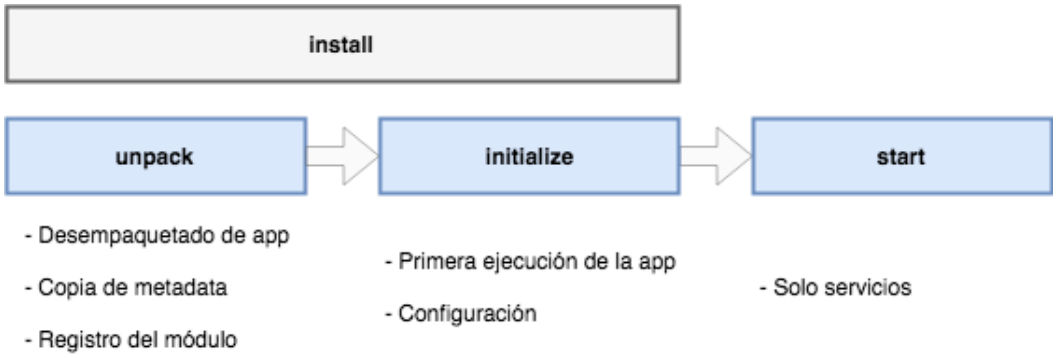


Figura 6-1. Ciclo de vida de un modulo de Nami.

6.2.1 Módulos de Nami

Cada uno de los paquetes software mencionados previamente se considera un módulo de *Nami*. Un módulo se compone de los ficheros ya compilados[41] junto con la lógica de configuración que a su vez puede hacer uso de ficheros auxiliares como librerías o plantillas de ficheros que se pueden renderizar (*Tabla 6-1*).

Tabla 6-1. Contenido de un módulo de Nami.

```
├── bitnami.json              # Metadata nami
├── COPYING                   # Información de licencias
├── files
|   └── wordpress             # Código compilado de WordPress
├── helpers.js
├── lib                       # Ficheros auxiliares de configuración
|   ├── component.js
|   ├── databases
|   ├── handlers
|   ├── host.js
|   ├── index.js
|   ├── log.js
|   ├── network.js
|   ├── templates
|   ├── utils.js
|   └── volume.js
├── main.js                   # Fichero principal de la lógica de configuración
└── templates                 # Plantillas renderizables
    ├── setup.php.tpl
    ├── upgradeDB.php.tpl
    ├── wordpress-https-vhost.conf.tpl
    └── wordpress-vhost.conf.tpl
```

[41] Por ficheros compilados aquí se entiende que están preparados para ser ejecutados en un entorno concreto. Por ejemplo, si se trata de una aplicación escrita en lenguaje C, compilar consiste en pasar de código fuente a código ejecutable.

6.3 Provisionado automático: Provisioner

Si bien *Nami* se encarga de la parte de configuración de una aplicación concreta, la herramienta *Provisioner* va más allá y se encarga de cualquier tipo de configuración que haya que realizar en una máquina para que esta pueda funcionar correctamente en la *cloud*. Estas tareas se pueden agrupar de la siguiente forma:

- Gestionar el ciclo de vida de los módulos de *Nami* (*Figura 6-1*). Provisioner se encarga de llamar a las operaciones *Nami unpack*, *initialize* y *start/stop* en el momento adecuado.
- Tareas de configuración no relacionadas directamente con las aplicaciones pero que son necesarias en una instancia de *cloud*. Ejemplos de estas tareas son configuración de parámetros del *kernel*, formateo de discos, obtención de direcciones IP de las interfaces de red, creación de scripts de servicio…
- Obtención de parámetros de usuario y comunicación con *CloudFormation* en general. Cuando se lanza una solución se especifican parámetros de configuración que el usuario final proporciona en tiempo de despliegue (Ver *Figura 5-2* del capítulo previo). Estos parámetros deben ser capturados por *Provisioner* de forma que cada uno de los módulos de *Nami* sea configurado en base a estos.

6.3.1 Introducción

Provisioner es un proyecto escrito en el lenguaje *Typescript*[42] (https://github.com/Microsoft/TypeScript) que es traducido a *JavaScript* para ser ejecutado. Algunos de sus componentes son:

- Una clase principal que implementa la lógica de tareas comunes.
- Definiciones que especifican como una solución debe ser desplegada.
- Clases para gestionar y personalizar tareas dependiendo de la plataforma. Por ejemplo, para instalar paquetes se utiliza *apt* en *Debian* y *yum* en *Centos*. Esto se consigue con herencia de clases y sobre escritura de métodos.
- Clases para gestionar y personalizar tareas relacionadas con las *clouds*. Por ejemplo, obtención de metadatos.
- Mecanismo para la implementación de "recetas" que se ejecutan en ciertos puntos del ciclo de vida del provisionado de las máquinas (*hooks*) si se cumplen unas determinadas condiciones.
- Definición de un conector para interactuar con *Nami*.

Tabla 6-2. Estructura general del código de Provisioner.

```
├── cloud
|   ├── base.ts
|   └── aws.ts
├── platform
|   ├── base.ts
|   ├── linux.ts
|   └── debian.ts
```

[42] Lenguaje de programación desarrollado por Microsoft considerado como un superconjunto de Javascript al que añade la noción de tipos a los objetos y variables.

```
├── provisioner.ts
├── recipe_runner.ts
├── recipes
|   ├── aws_cfn_signal.ts
|   ├── bitnami_agent.ts
|   ├── data_disk.ts
|   ├── shared_disk.ts
|   ├── ssh_settings.ts
|   ├── system_packages.ts
|   └── welcome_message.ts
└── stack_definition.ts
```

6.3.2 Implementación de la cloud AWS en Provisioner

Para dar soporte a *WordPress High Availability* en *Provisioner* se debe llevar a cabo una adaptación de *Provisioner* a la *cloud AWS*. Para ello, y gracias a la jerarquía orientada a objetos del código de la herramienta, se crea una clase que incluye los métodos necesarios para la interacción con la *cloud*. Los métodos implementados se describen a continuación. Sirven para obtener *metadatos* y *user-data*.

6.3.2.1 _getUserDataScript, _getUserDataScriptNow

- Argumentos: -
- Devuelve: El script de *user-data* definido en las plantillas *CloudFormation*.

Esta pareja de métodos descarga el script de *user-data*. Se accede al script a través de la *API* de *AWS* http://169.254.169.254/latest/user-data. Existen dos versiones de esta funcionalidad (con y sin el sufijo *Now*). El objetivo de esta separación es utilizar una propiedad de la clase como caché de esta información. Llamando a *_getUserDataScript*, solo se llamará a *_getUserDataScriptNow* si la propiedad *_userDataScriptOnce* no está definida. Es el segundo método el que realmente realiza la petición a la *API*.

Para ver el resultado devuelto (Figura 6-2), se puede ejecutar *Provisioner* con el comando *eval* que permite invocar los métodos de las clases.

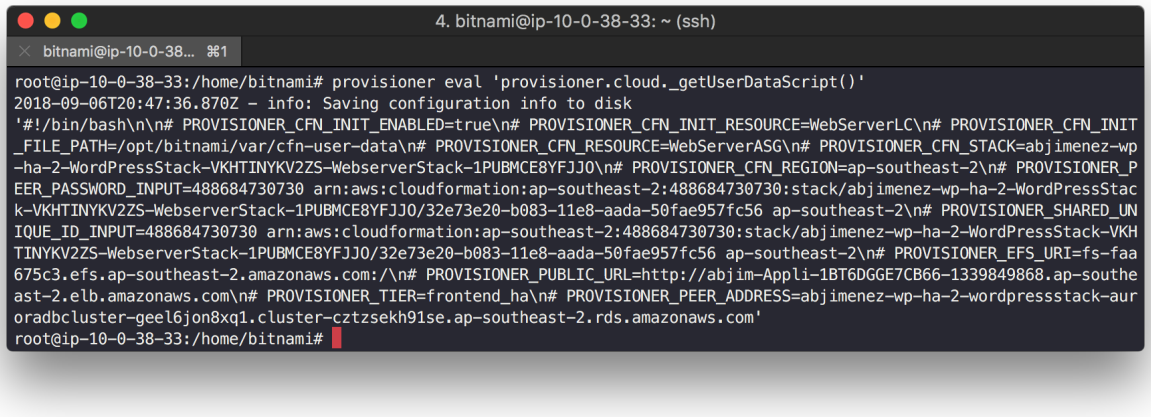


Figura 6-2. *Ejecución del método de Provisioner _getUserDataScript.*

6.3.2.2 _getUserDatatNow

- Argumentos: Nombre de una variable de *user-data.*
- Devuelve: El valor de esa variable.

Si los métodos anteriores devuelven la totalidad del script de *user-data*, este método devuelve el valor concreto de una de las variables definidas. Un método definido en la clase padre se encarga de parsear el contenido para obtener el valor correcto.

En la *Figura 6-3* se puede ver como llamando al método *getUserData* de la clase base padre, este llama a _getUserDataNow de la clase *AwsCloud* y se obtiene el correspondiente valor.

Figura 6-3. Ejecución del método de Provisioner getUserData.

6.3.2.3 _getMetadataNow, _getInstanceIdentityDocument, _getInstanceIdentityDocumentNow

- Argumentos: Nombre de una variable de metadata.
- Devuelve: El valor de esa variable.

Además de *user-data*, existe otro tipo de metadatos o información relativa al despliegue realizado, bajo la *API* http://169.254.169.254/latest/dynamic/instance-identity/document. En concreto, la propiedad *account-id* es de utilidad para poderse incorporar a los datos recolectados por el agente de estadísticas. Esta propiedad indica la cuenta de *AWS* desde la que se ha lanzado la solución. De esta forma, se puede conocer el número de despliegues que hace un mismo usuario.

Es importante destacar que el identificador de la cuenta no se envía de forma directa (*Figura 6-4*) puesto que es un dato sensible y por ello se procesa previamente utilizando el algoritmo de *hash SHA256*. Este algoritmo se caracteriza por dar siempre la misma salida ante la misma entrada, pero no es posible obtener la entrada conociendo únicamente la salida.

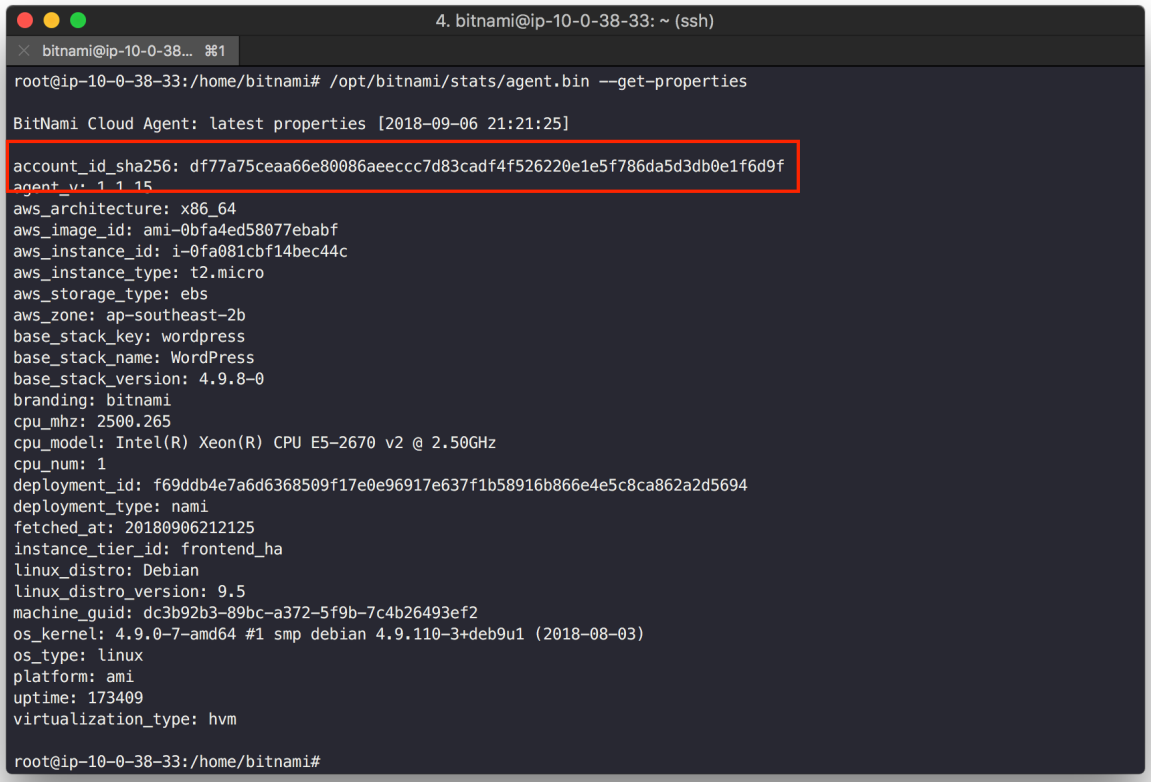


Figura 6-4. Información recolectada por el agente de estadísticas.

Siguiendo la misma idea de cachear la información solicitada de forma remota (a la *API*) se han definido métodos con y sin el sufijo *Now*.

La *Figura 6-5* muestra el valor devuelto cuando se invocan los métodos *_getMetadataNow* con el argumento *account-id* y el valor devuelto cuando se invoca *_ getInstanceIdentityDocument*.

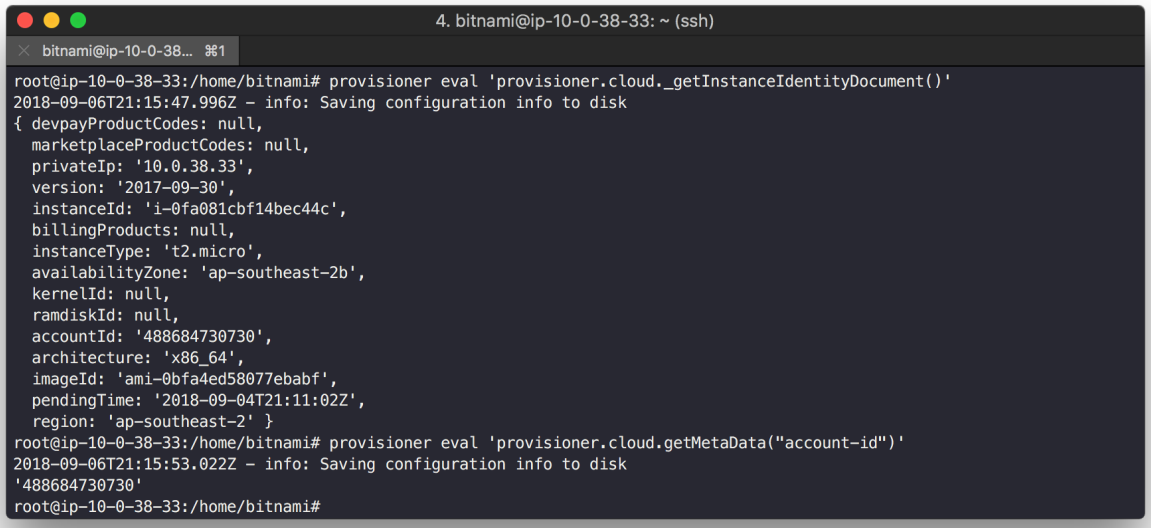


Figura 6-5. Ejecución de los métodos de Provisioner _getInstanceIdentityDocument y getMetaData.

Existen otras muchas modificaciones que se le aplican a *Provisioner* para ser capaces de dar soporte a *WordPress High Availability*. Algunos de estos cambios son explicados en la sección 8.3.6, ya que se requiere conocer el contexto por el que esas modificaciones son necesarias.

El código implementado se puede ver en el Anexo A: Código fuente implementado de Provisioner.

6.3.3 Definición de la información de la solución

Como se especifica en la sección 6.3.1, para poder invocar los métodos de *Nami* con los parámetros adecuados y para saber que paquetes hay que incluir en las imágenes de *cloud*, es necesario que se especifiquen en algún punto. Para ello se utiliza lo que se conoce como *Provisioner definition.*

Es un fichero *YAML* que contiene información relativa a los componentes que deben ser incluidos en las imágenes de cloud. Esta información consiste en módulos de *Nami* y sus dependencias (paquetes del sistema). Puesto que las dependencias pueden variar entre distintas versiones de las aplicaciones, estas no se dejan indicadas en la definición. Durante el proceso de construcción de las imágenes se consulta una base de datos y se inyectan en el fichero los paquetes que son necesarios.

La definición también incluye otro tipo de información de configuración de la máquina como por ejemplo los puertos que deben estar abiertos o los parámetros que los módulos de Nami van a recibir en su inicialización. Por ejemplo, *WordPress* recibe las credenciales de la base de datos. Como *Provisioner* es quien llama a *Nami*, *Provisioner* debe conocer los valores de estos parámetros. Para conocerlos, los obtiene de la *user-data* que *CloudFormation* elabora. En la *Figura 6-6* se muestra un esquema con el viaje que realizan los parámetros.

También es importante destacar que esta definición está organizada por medio de lo que se denominan *tiers*. Cada *tier* representa un tipo de instancia en una topología concreta. Una misma definición puede servir para varias *clouds*, como es el caso. Cada *cloud* define una topología

(para soluciones con varios nodos) que indica el tipo de *tiers* que debe incluir. Para *Wordpress*, *Google* utiliza la topología "*frontend_database*". Esta topología incluye las tiers *frontend* y *database*. En *AWS* en cambio, se define la topología "*high_availability*" y solo utiliza la tier "*frontend_ha*" puesto que para la base de datos no se utiliza una instancia sino el servicio *RDS*.

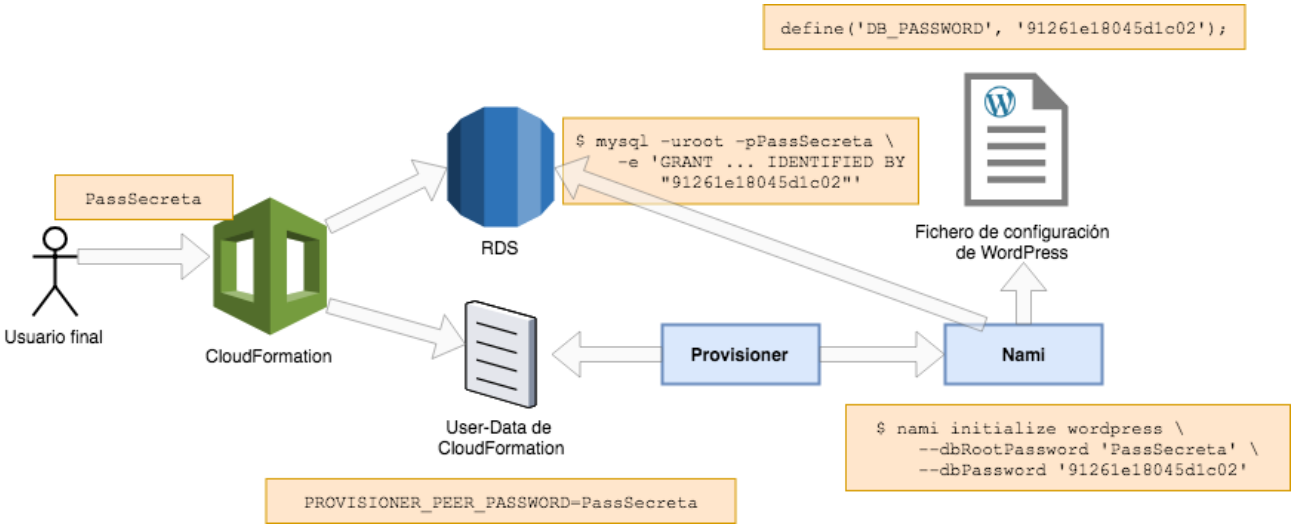


Figura 6-6. Viaje de parámetros desde el usuario a los servicios.

La definición completa puede encontrarse en el *Anexo B: Contenido de la definición de Provisioner*.

6.4 Comunicación entre *CloudFormation* y las instancias

Entre *CloudFormation* y las instancias debe haber cierta señalización para tareas como obtención de parámetros introducidos por el usuario final en tiempo de despliegue, confirmación de éxito en la inicialización de las instancias o consulta de metadatos del despliegue.

6.4.1 cfn-tools

Se trata de un conjunto de herramientas escritas en *Python* que ayudan a la comunicación entre la instancia y *CloudFormation* (en esa dirección, desde la instancia, se consulta a *CloudFormation*). Permiten instalar software o manejar los servicios. [48]

Algunas de las herramientas utilizadas en este proyecto son:

6.4.1.1 cfn-init

Permite ejecutar scripts o copiar ficheros que se pueden definir desde las plantillas de *CloudFormation* mediante el atributo *AWS::CloudFormation::Init*. Por ejemplo, en *WordPress High Availability* se utiliza para el paso de secretos, es decir, los parámetros de usuario que son contraseñas (*Tabla 6-3*). Si desde la instancia se ejecuta el comando *cfn-init* mostrado en la *Figura 6-7*, se crea un fichero donde se indica en la plantilla de *CloudFormation*.

Tabla 6-3. Bloque CloudFormation Init para paso de secretos.

```
"AWS::CloudFormation::Init" : {
    "config": {
        "files": {
            "/opt/bitnami/var/cfn-user-data": {
                "content": {
                    "Fn::Join": [
                        "",
                        [
                            "\n# PROVISIONER_PEER_PASSWORD=",
                            { "Ref": "DBMasterUserPassword" },
                            "\n# PROVISIONER_APP_PASSWORD=",
                            { "Ref": "WordpressAdminPassword" }
                        ]
                    ]
                },
                "mode": "000400",
                "owner": "root",
                "group": "root"
            }
        }
    }
}
```

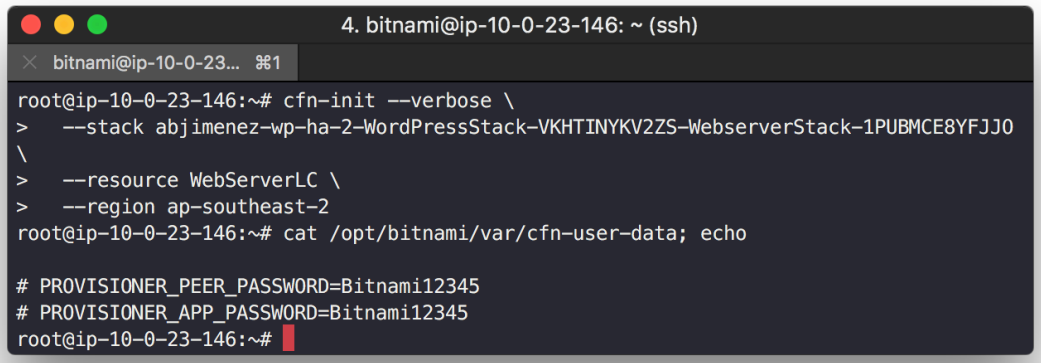


Figura 6-7. Ejecución de cfn-init.

6.4.1.2 cfn-signal

cfn-signal es una herramienta que se parece bastante a *cfn-init*. La utilidad de esta es la de enviar una señal de confirmación a *CloudFormation* sobre el estado tras la inicialización. Para mostrar su uso, en la *Tabla 6-4* se puede observar la receta de *Provisioner* que se encarga de realizar esta llamada. Cuando la llamada es recibida, se muestra como un evento en el panel de control de *CloudFormation* (*Figura 6-8*).

Tabla 6-4. Extracto de receta de Provisioner para el envío de señal.

```
    const resource: string = await cloud.getUserData("PROVISIONER_CFN_RESOURCE");
    const stack: string = await cloud.getUserData("PROVISIONER_CFN_STACK");
    const region: string = await cloud.getUserData("PROVISIONER_CFN_REGION");
    const signal: number = (input.eventName === "afterFirstboot") ? 0 : 1;
```

```
    if (resource && stack) {
      logger.info(`Notifying boot status from ${input.eventName}, code ${signal}`);
      try {
        $os.runProgram("env", [
          "PYTHONPATH=/usr/lib/python2.7/site-packages", "cfn-signal",
          "-e", signal, "--stack", stack, "--resource",  resource, "--region", region
        ]);
      } catch (e) {
        ...
```

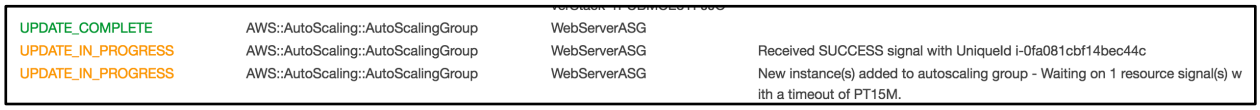


Figura 6-8. Recepción de la señal SUCCESS en el panel de control de CloudFormation.

6.4.2 *cloud-init* y *user-data*

Si la comunicación instancia – CloudFormation se realiza con *cfn-tools*, la comunicación *CloudFormation* – instancia se hace mediante el uso de *cloud-init* y lo que se llama *user-data.*

cloud-init (https://cloud-init.io/) es un proyecto colaborativo para establecer una forma común de inicializar las instancias de *cloud* independientemente del proveedor. AWS lo usa para ejecutar durante la inicialización de la instancia el script definido en la propiedad *UserData* de las instancias (*Tabla 6-5*). Otras *clouds* también utilizan esta misma herramienta, entre ellas *Azure*, *VMware* o *Google Cloud Platform*. [49]

Sin embargo, en el caso de *WordPress High Availability*, se accede directamente al contenido de *user-data* puesto que no es necesario ejecutar ningún script. El campo de *user-data* es utilizado simplemente para definir variables de entorno que *Provisioner* es capaz de leer utilizando un método específico para parsear el contenido. Estas variables permiten personalizar el comportamiento de *Provisioner* desde la plantilla de *CloudFormation* y también pasar información de usuario.

Tabla 6-5. Uso de user-data en la solución.

```
"UserData": {
    "Fn::Base64": {
        "Fn::Join": [
            "",
            [
                "#!/bin/bash\n",
                "\n# PROVISIONER_CFN_INIT_ENABLED=true",
                "\n# PROVISIONER_CFN_INIT_RESOURCE=WebServerLC",
                "\n# PROVISIONER_CFN_INIT_FILE_PATH=/opt/bitnami/var/cfn-user-data",
                "\n# PROVISIONER_CFN_RESOURCE=WebServerASG",
                "\n# PROVISIONER_CFN_STACK=", { "Ref": "AWS::StackName" },
                "\n# PROVISIONER_CFN_REGION=", { "Ref": "AWS::Region" },
                ...
```

Este acceso directo al contenido de *user-data* se hace realizando una petición a la *API* de *AWS* (*Figura 6-9*). Con esta *API*, también es posible acceder a campos de metadatos.

Figura 6-9. Acceso a user-data mediante API de AWS.

6.5 Diagrama de componentes

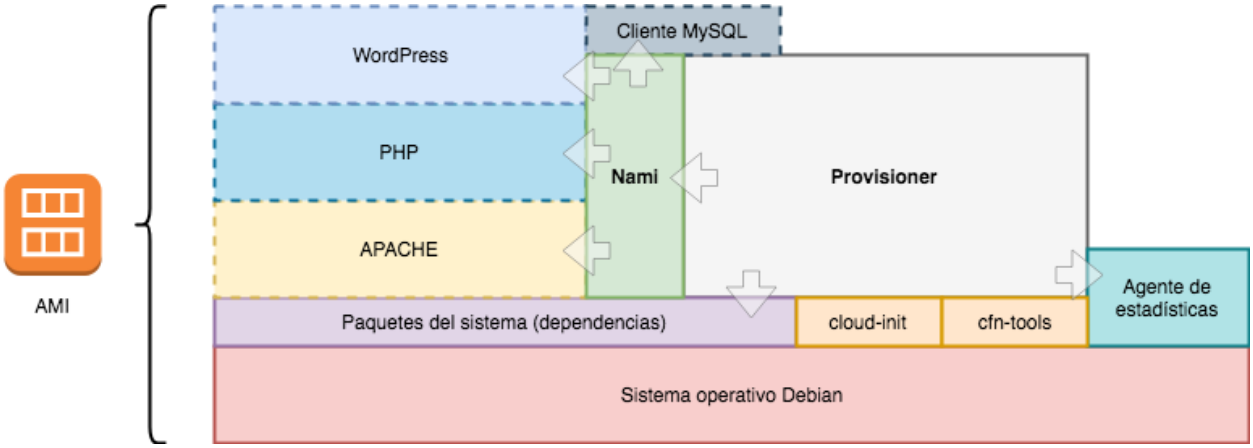


Figura 6-10. Diagrama de bloques que componen la AMI.

A modo de resumen de este capítulo, en el diagrama mostrado en la *Figura 6-10*, se pueden observar los distintos elementos que componen la imagen (*AMI*) de *WordPress High Availability*. Empezando por los componentes más básicos, en la parte baja, se encuentran el sistema operativo seguido de las dependencias (paquetes del sistema) de los módulos de *Nami* y las herramientas de configuración.

También se encuentran las herramientas *cloud-init* y *cfn-tools*, necesarias para la comunicación entre la instancia y *CloudFormation,* así como el agente para recolección de estadísticas sobre el uso de la solución.

En la parte central, se observan las herramientas de configuración *Nami* y *Provisioner*. Por último, las piezas más importantes: los módulos de *Nami en línea discontinua*. Son 4, *Apache*, *PHP* y *WordPress* por un lado y un cliente *MySQL* que *WordPress* requiere para las conexiones con la base de datos.

Mediante flechas, se pretende indicar qué elementos se encargan de configurar y controlar a qué otros elementos. Así, *Provisioner* es quien se encarga de gestionar la instalación de paquetes del sistema de los que dependen los módulos de *Nami*, se encarga de configurar el agente de estadísticas e indirectamente, a través de la herramienta *Nami*, configurar e iniciar los módulos de las aplicaciones.

7 INTEGRACIÓN DE LA SOLUCIÓN EN LOS PROCESOS DE BITNAMI

El mantenimiento de los productos que la compañía publica forma parte del valor que *Bitnami* aporta. Sin ir más lejos, un ejemplo claro de estas tareas de mantenimiento se dio durante el incidente de seguridad conocido como *L1 Terminal Fault* [50] afectando el *kernel* de *Linux*.

Ante esta situación cada una de las distribuciones deben hacer pública una nueva versión del kernel que solucione la vulnerabilidad. Es entonces cuando *Bitnami* debe aplicar ese parche a cada una de las soluciones proporcionadas en forma de imágenes de *cloud*.

Cada uno de los procesos de los productos que *Bitnami* provee debe estar automatizado (construcción, testeo y publicación) puesto que, de no ser así, el parcheo de imágenes pasaría de ser una tarea de pocas horas a ser una tarea de varias semanas. En términos de automatización *WordPress High Availability* no es una excepción.

7.1 Construcción y despliegue

En esta solución se pueden observar dos elementos de construcción: las *AMIs* y las plantillas de *CloudFormation*.

Respecto a la construcción de imágenes, en *Bitnami* ya existen procesos bien definidos puesto que esta tarea no cambia respecto a la multitud de imágenes que se mantienen. Si es cierto que con soluciones que utilizan la herramienta *Provisioner*, se deben realizar modificaciones en el proceso[43].

En cuanto a las plantillas de *CloudFormation*, para otras *clouds*[44], son generadas. Para ello, se utiliza el proyecto *BRadmin* ya mencionado, en concreto el sistema de renderizado *ERB* [51] de *Ruby*. El proceso de renderizado se esquematiza en la *Figura 7-1*. Esta forma de trabajar con las

[43] Estas modificaciones están fuera del alcance del proyecto puesto que lo ha realizado un equipo diferente.
[44] Por ejemplo, la implementación de las soluciones multi-nodo de OCI https://github.com/bitnami/oci-multi-tier.

plantillas tiene numerosas ventajas, como por ejemplo que se puedan definir propiedades comunes a muchas soluciones y mediante código, se insertan en las plantillas.

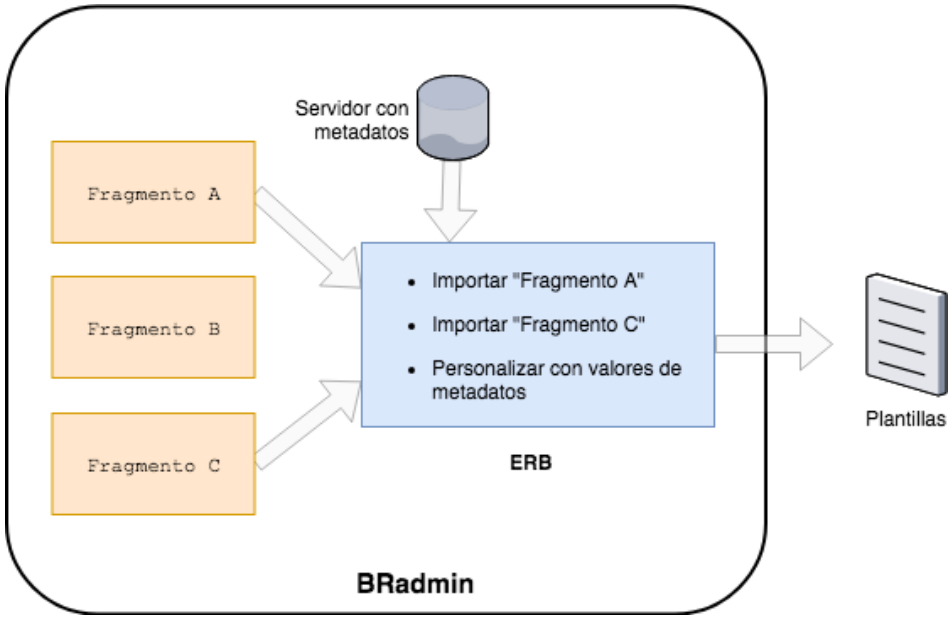


Figura 7-1. Generación de plantillas mediante BRadmin y ERB.

Sin embargo, siguiendo este método, se requiere que *Bitnami* sea el mantenedor único de las soluciones ya que el código de las plantillas previamente a ser renderizadas no se comparte, únicamente el resultado de este renderizado, por lo que cambios aplicados en la salida del fichero son relativamente difíciles de incorporar en el código original.

Precisamente por este motivo, no es una solución válida para el caso de *WordPress High Availability* puesto que el equipo de *AWS Quick Start* también contribuye con cambios en las plantillas para su mantenimiento. Ante esta situación, se debe implementar una forma diferente de trabajar con las plantillas. Para este caso, la "fuente de verdad" reside en el repositorio compartido en *GitHub* y mediante *Jenkins* y *BRadmin* se realiza una descarga de las plantillas y una preparación mediante modificaciones menores para su posterior testeo.

Para la gestión de esta nueva forma de trabajo con plantillas se ha definido un nuevo paso en la ejecución de trabajos de *Jenkins* y un nuevo módulo de *BRadmin* que recoge toda la lógica necesaria. Puesto que *BRadmin* es un proyecto de una gran envergadura es necesaria su modularización para que continúe siendo mantenible.

7.1.1 Modificación de la configuración de Jenkins

Jenkins es un proyecto de software libre escrito en *Java* que se utiliza para automatizar tareas de construcción, testeo y publicación de software [52]. En *Bitnami*, sin embargo, *BRadmin* es quien tiene la lógica para realizar este tipo de acciones y *Jenkins* se utiliza como una interfaz para controlar y visualizar la construcción de artefactos, su despliegue y testeo, ya sea de forma manual o automática (*Figura 7-2*).

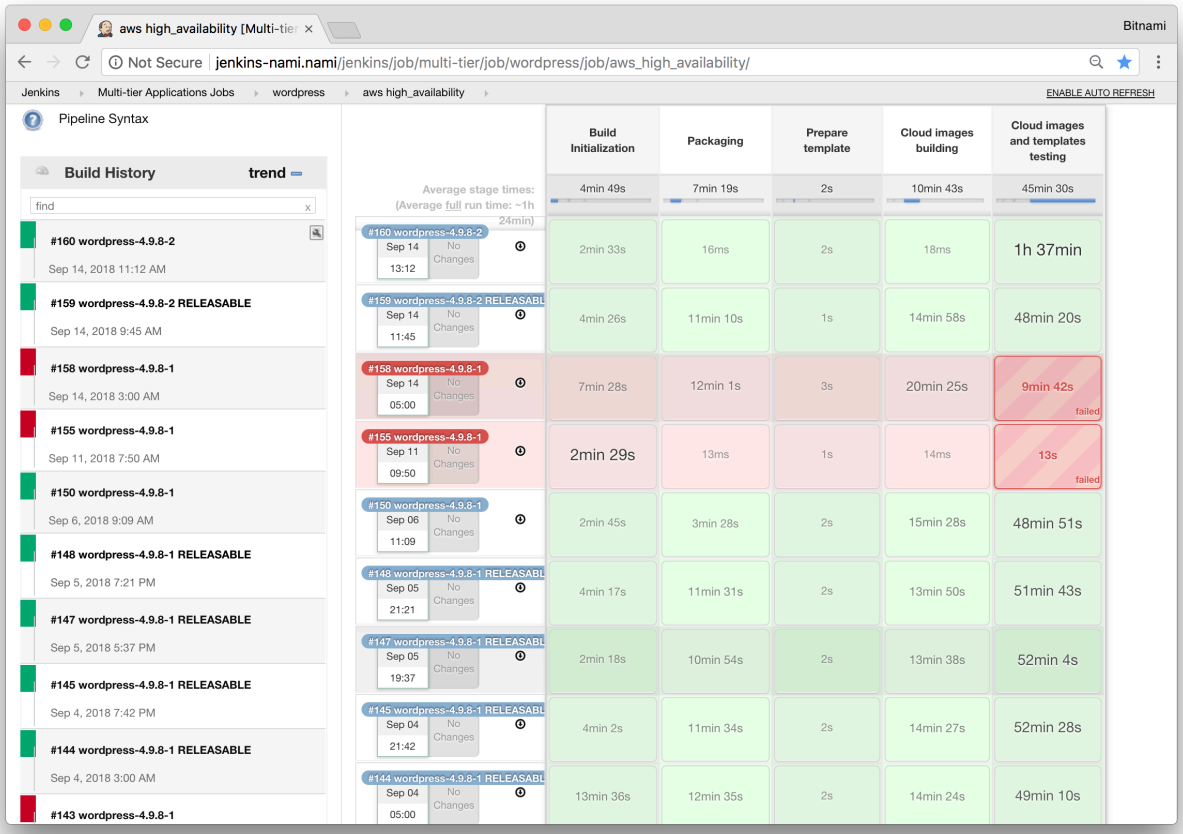


Figura 7-2. Vista de un trabajo de Jenkins ejecutado diariamente para la construcción y testeo de la solución Wordpress High Availability.

Puesto que para la solución *WordPress High Availability* es necesaria la descarga de las plantillas de forma externa, se abre una nueva gama de posibilidades a tener en cuenta cuando se quiere construir y testear la solución. Las variables a tener en cuenta son:

- Reutilización de AMI creada anteriormente.
- Utilización de la AMI especificada en las plantillas.
- Testeo de cambios en las plantillas.

Para controlar estas casuísticas se definen tres nuevos parámetros de entrada en los trabajos. Estos se definen en un fichero *XML*[45] (*Tabla 7-1*). Se pueden especificar condicionales mediante el uso de *handlebars*[46] (https://handlebarsjs.com/) que personalicen los parámetros para cada uno de los trabajos. El resultado puede verse en la *Figura 7-3*.

Tabla 7-1. Definición de parámetros de entrada de trabajos de Jenkins.

```
{{#if dependsOnExternalRepository}}
        <hudson.model.StringParameterDefinition>
          <name>REPOSITORY_URL</name>
          <description>Repository url for applications using templates from an
external git server.</description>
          <defaultValue>github.com/bitnami/quickstart-bitnami-wordpress</defaultValue>
```

[45] eXtensible Markup Language o lenguaje de marcado ampliable.
[46] Handlebars es un sistema de renderizado de plantillas para el lenguaje JavaScript. En cierto modo, es similar al preprocesado de código C.

```
        </hudson.model.StringParameterDefinition>

        <hudson.model.StringParameterDefinition>
          <name>REPOSITORY_BRANCH</name>
          <description>Repository branch.</description>
          <defaultValue>master</defaultValue>
        </hudson.model.StringParameterDefinition>

        <hudson.model.BooleanParameterDefinition>
          <name>USE_TEMPLATE_AMI</name>
          <description>Use the AMI specified in the template and don't perform build.
If REUSE_BUILD is specified, the AMI from the referenced job will be
used.</description>
          <defaultValue>false</defaultValue>
        </hudson.model.BooleanParameterDefinition>
{{/if}}
```

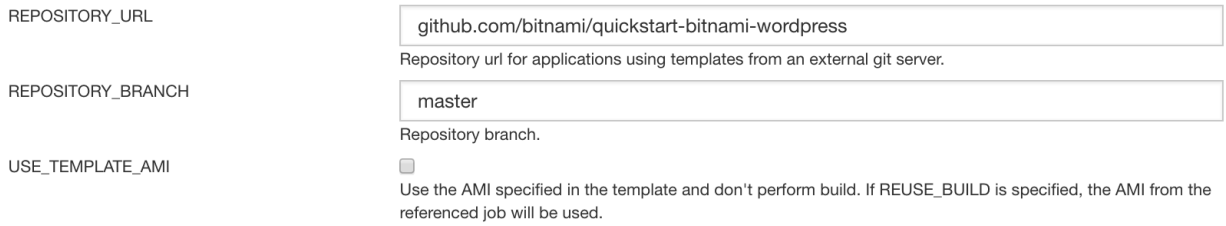


Figura 7-3. Parámetros de entrada en el trabajo de Jenkins.

Además de definir parámetros de entrada, también es necesario obteneros y realizar acciones en base a los valores de estos. Para especificar la lógica de esas acciones se utiliza el lenguaje *Groovy*[47] (http://groovy-lang.org/). Por mencionar algunos de los cambios introducidos e ilustrar la sintaxis de *Groovy,* en la *Tabla 7-2* se muestra la definición de un método que clona un repositorio de *git*. Este a su vez llama a otro método denominado *withCredentials* que obtiene secretos configurados en *Jenkins* como por ejemplo credenciales de tipo usuario/contraseña.

Este método es invocado desde la definición de las fases de cada trabajo (*Figura 7-4*). El paso de *preparación de la plantilla* ha sido introducido para esa solución.

Tabla 7-2. Definición de un método que clona un repositorio de git.

```
def clonePrivateGitHubRepository(repo_url, branch = 'master', pull_request = '',
credentialsId = 'github-bot') {
    withCredentials([
        usernamePassword(credentialsId: credentialsId, usernameVariable:
'GITHUB_USER', passwordVariable: 'GITHUB_PASSWORD')
    ]) {
        sh "git clone -b ${branch}
\"https://${GITHUB_USER}:${GITHUB_PASSWORD}@${repo_url}\" ."
        if (pull_request != '') {
            sh "git fetch origin refs/pull/${pull_request}/head:pr-${pull_request}"
            sh "git checkout pr-${pull_request}"
        }
    }
```

[47] Groovy es un lenguaje de programación desarrollado por la fundación Apache compatible con Java.

```
}
```

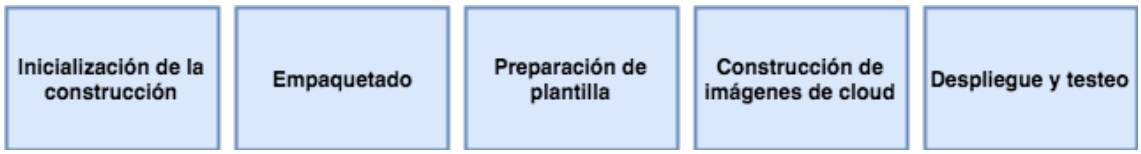


Figura 7-4. Fases por las que pasa un trabajo de Jenkins.

Las modificaciones realizadas en la configuración de Jenkins se pueden ver en el *Anexo E: Configuración de Jenkins.*

7.1.2 Módulo de BRadmin

Además de cambios en la configuración de *Jenkins*, han sido necesarios cambios en *BRadmin*, previamente mencionado. Las nuevas funcionalidades que son necesarias para poder realizar un despliegue de esta solución para su posterior testeo se han incluido como un módulo de *Ruby*. Crear módulos independientes para cada funcionalidad ayuda al mantenimiento de un gran repositorio como es *BRadmin*.

En el *Anexo F: Configuración de BRadmin*, se puede ver la definición completa del nuevo módulo. A continuación, se describen algunos de los métodos y funcionalidades más relevantes.

7.1.2.1 prepare_template

Es el método principal del módulo. Tras obtener las plantillas descargadas de *GitHub*, este método, con ayuda de otros métodos auxiliares, realiza las modificaciones necesarias para poder inyectar configuraciones que sean convenientes. Por ejemplo, cuando se construye una nueva *AMI* y esta se quiere testear, se debe indicar el identificador de imagen en el recurso *LaunchConfiguration* del grupo de escalado automático. Esto se muestra en la *Tabla 7-3*.

Tabla 7-3. Modificación de las plantillas para inyección de AMI construida.

```
find_built_image
if image_id.present?
  templates[:webserver] = set_template_resource_property(
     templates[:webserver],
     'WebServerLC',
     'ImageId',
     image_id
  )
end
```

7.1.2.2 upload_template_for_test_deployment

Se trata de un método auxiliar que es necesario para subir las plantillas tras ser modificadas. La nueva *URL* se inyecta en otras plantillas para que sean importadas. Es necesario modificar, subir y referenciar de forma ordenada, es decir, la plantilla padre es la última en modificarse para que importe todas las demás plantillas ya modificadas.

7.1.2.3 set_deletion_policy

Modifica la política por defecto de ciertos recursos que crean copias de respaldo cuando son eliminados. Este tipo de copias no son convenientes en un entorno de integración continua.

7.1.2.4 Utilización de la consola de BRadmin

Para la creación de nuevos métodos en BRadmin, es muy útil el uso de la consola de desarrollo en la que se pueden instanciar las clases definidas y se puede invocar a los métodos. Un ejemplo de esto se ilustra en la *Figura 7-5*. Para comenzar, se instancia la clase *AWS cloud*. Esta clase define un método *find_deployment* que devuelve un objeto correspondiente a un despliegue de *CloudFormation* concreto. En esta nueva clase se define un nuevo método *load_balancers*, de utilidad para poder obtener el nombre de dominio y así poder ejecutar tests en la interfaz web.

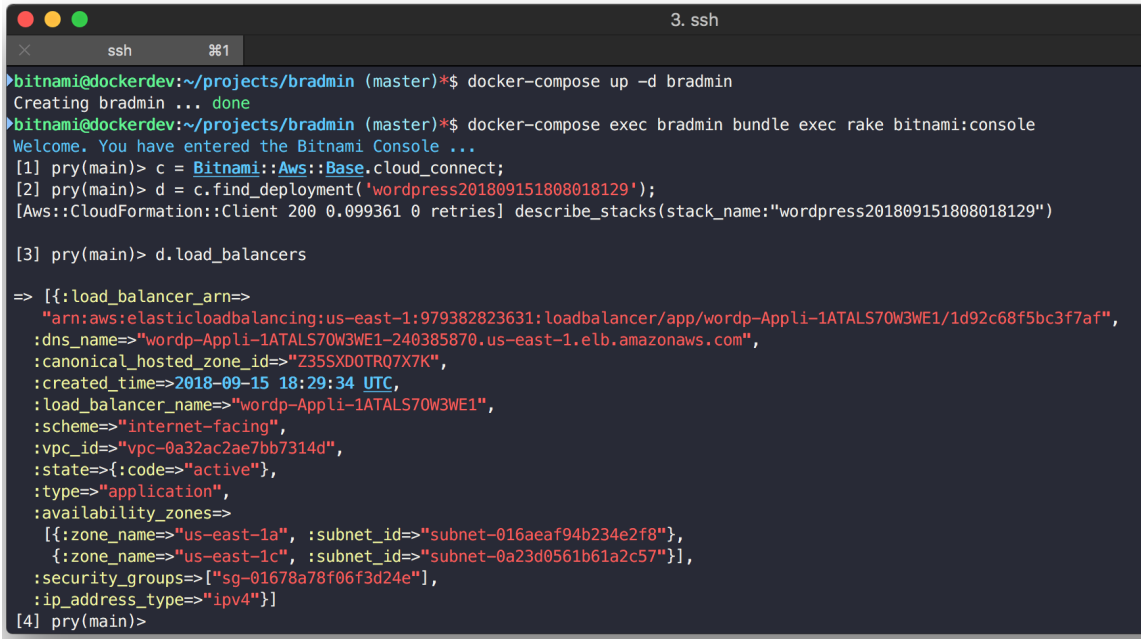


Figura 7-5. Utilización de la consola de BRadmin para la ejecución del método load_balancers.

7.2 Testeo

Cuando las plantillas están preparadas, se puede invocar a un método de *BRadmin* que lanza la solución con *CloudFormation*. De esta manera, sobre el despliegue se pueden realizar pruebas que verifican que el correcto funcionamiento de la solución.

Gracias a las pruebas realizadas se detectan fallos introducidos por cambios realizados en las plantillas o en cualquiera de los componentes de la *AMI*. Además, la solución se testea diariamente, por lo que también se detectan cambios producidos en los servicios de *AWS* y que requieren actualizaciones de las plantillas o en los procesos de construcción de la imagen.

Las pruebas realizadas se pueden englobar en dos categorías: tests funcionales (o de aplicación) y tests de verificación (o de validación) que se describen a continuación.

7.2.1 Tests funcionales

Por un lado, están los tests funcionales. Son pruebas que chequean la solución desde el punto de vista del usuario final. Por ello, se realizan pruebas contra la interfaz web que es precisamente a lo que el usuario tiene acceso. Como su nombre indica, se testean las funcionalidades de la solución.

La tecnología utilizada para este tipo de pruebas es *CasperJS* (http://casperjs.org/) una plataforma de testeo para *JavaScript* que utiliza navegadores web sin interfaz gráfica, lo que se conoce como *headless browsers*.

CasperJS permite navegación a través de una web de forma programática. Permite hacer clic en enlaces, rellenar formularios, descargar recursos y otras muchas acciones. Para indicar los elementos en los que actuar se utilizan selectores *CSS*[48]. Además de navegación, *CasperJS* también ofrece métodos para realizar capturas de pantalla y realizar comprobaciones acerca del contenido esperado frente al contenido real.

Para WordPress High Availability, se ejecutan un total de 27 tests, realizando en conjunto 287 comprobaciones. (*Figura 7-6*).

```
CaspRun INFO  PASS Auto TLS switch is visible
CaspRun INFO  PASS SMTP Auth switch is visible
CaspRun INFO  PASS Submit button is visible
CaspRun INFO  PASS Settings saved
CaspRun INFO  PASS Recipient field exists
CaspRun INFO  PASS Submit button is visible
CaspRun INFO  PASS Success message exists
CaspRun INFO  PASS Email was successfully sent
CaspRun INFO  PASS User bar exists
CaspRun INFO  PASS Logout button exists
CaspRun INFO  PASS Logged out successfully
CaspRun INFO  Clearing client scripts to be injected
CaspRun INFO  PASS 287 tests executed in 268.248s, 287 passed, 0 failed, 0 dubious, 0 skipped.
CaspRun INFO  Result log stored in /home/agent/workspace/multi-tier/wordpress/aws_high_availability/161/test-workspace/aws-
template/bts/functional/WordPress/reports/functional-applicationTests.xml
bts     INFO  Finished running tests for stage 'applicationTests'. exitCode = 0
```

Figura 7-6. Resultado mostrado tras la ejecución de tests funcionales.

Para mostrar la sintaxis de *CasperJS*, en las *Tabla 7-4* y *Tabla 7-5* se muestra uno de los tests añadidos para esta solución.

Tabla 7-4. Ejecución de test que añade un comentario a una entrada del blog.

```
casper.test.begin('Add comment', function(test) {

casper.start(`http://${casper.urlData.host}:${casper.urlData.port}${casper.urlData.pre
fix}`);
  casper.login(casper.defaultUser, test);
  casper.visitSite(casper.urlData, test);
  casper.goToPublishedPost(casper.testDefinitions.posts[0], test);
  casper.addComment(casper.testDefinitions.comments[0], test);
  casper.run(function() { test.done(); });
```

[48] Cascading Style Sheets, es un lenguaje que permite la definición del formato de presentación de elementos HTML.

```
});
```

Tabla 7-5. Función auxiliar para añadir comentario.

```
casper.addComment = function addComment(text, test) {
  casper.waitUntilVisible('textarea#comment', function() {
    test.assertVisible('textarea#comment', 'Comment input exists');
    this.captureScreen('commentInput');
    casper.echo(`Adding comment content "${text}"`);
    this.sendKeys('textarea#comment', text);
  });
  casper.waitUntilVisible('input#submit', function() {
    test.assertVisible('input#submit', 'Submit button exists');
    this.captureScreen('submitButton');
    this.click('input#submit');
  });
  casper.waitUntilVisible(x(`//div[@class="comment-content"]/p[.="${text}"]`),
function() {
    test.assertVisible(x(`//div[@class="comment-content"]/p[.="${text}"]`), 'Comment
added successfully');
    this.captureScreen('commentAddedSuccessfully');
  });
};
```

Los tests añadidos a los ya existentes se muestran en el *Anexo G: Tests Funcionales*.

7.2.2 Tests de verificación

Además de los tests funcionales, se realizan tests de verificación, también llamados de validación. Con ellos se pretende realizar chequeos sobre la configuración de la solución que van más allá de la funcionalidad en sí misma. Por ejemplo, se definen tests para comprobar los permisos de los ficheros o la presencia de mensajes de error en los ficheros de *logs*. Para realizar estos tests se utiliza la tecnología *Mocha* (https://mochajs.org/).

Mocha es un entorno de testeo para *JavaScript* que permite organizar las pruebas que se deseen implementar por medio de llamadas a funciones (como *describe* o *it*). Soporta la ejecución de código asíncrono[49] y también implementa métodos para reintentar tests fallidos. Para realizar las comprobaciones se utilizan librerías adicionales como *Chai* (https://www.chaijs.com/).

Además de tests básicos como los mencionados anteriormente, para esta solución se desea hacer un chequeo adicional del funcionamiento de recursos como *EFS* o el balanceador de carga. Para ello, se ha definido una librería de tests (*remote.js*) que permite la ejecución de comandos y tests de forma remota, para así poder verificar el correcto funcionamiento de estos recursos. En el *Anexo H: Tests de verificación* se muestra el contenido de esta librería, así como los ficheros de tests *shared_disk.js* y *load_balancer.js*, donde se puede apreciar la sintaxis de *Mocha*. A continuación, se explican algunas de las ideas que están detrás de la implementación de dichos tests.

[49] Se denominan código asíncrono a una forma de ejecución de código que no es secuencial. En este paradigma, cuando se llama a una función, no se espera a que esta termine y se continúa en el flujo de Código. La función eleva un evento de terminación que se debe capturar.

7.2.2.1 Test de sistema de ficheros compartido

Para la ejecución del test de *EFS*, por un lado, se comprueba que exista una entrada de montado de dispositivo en el fichero */etc/fstab* y que se encuentra actualmente montado (con el comando *mount -l*). Por otra parte, también se crea un fichero en el punto de montaje de un nodo y se comprueba que ese fichero es accesible desde los demás nodos.

En la *Tabla 7-6* se muestra parte de la implementación descrita. También se puede observar el uso de las funciones *describe*, *it* y *expect* de *Mocha* y *Chai*. Son funciones que aportan cierta legibilidad al código. También se puede observar el uso de la función *remote.runProgram*, que ejecuta un comando en una máquina remota.

Tabla 7-6. Test de verificación para el chequeo de EFS.

```
  describe('Replication of files', function() {
    _.each(nodes, (node) => {
      it(`node ${node} should replicate the file`, function() {
        this.timeout(5000);
        this.retries(4);
        $util.sleep(1);
        const content = remote.runProgram(node, 'cat', $file.join(mount.point,
testFile));
        expect(content).to.contain(fileContent);
      });
    });
  });
});
```

7.2.2.2 Test de balanceador de carga

Otro de los recursos que resulta interesante chequear es el balanceador de carga. Para ver si el balanceador distribuye las peticiones, se ha optado modificar *Apache* de forma que este envíe una cabecera *HTTP* con el valor de la *IP* de la instancia (variable *SERVER_ADDR*) mediante el uso de la directiva *Header* [53]. Esta modificación es realizada por el test en sí. Tras el cambio, se reinicia el servicio y se realizan solicitudes al balanceador de carga.

Las respuestas se analizan para obtener el valor de la cabecera esperada y se comprueba que todos los nodos hayan enviado una respuesta. Esto funciona ya que las peticiones consecutivas que se envían no mantienen la *cookie* de sesión *AWSALB (sticky session)*. Tras la comprobación, se restaura el fichero de configuración de *Apache* a su valor original.

Este test utiliza las funciones de *Mocha before*, *beforeEach* y *after*, que permiten realizar acciones antes y después de la ejecución de un determinado test.

La salida de la ejecución de estos tests se puede observar en la *Figura 7-7*.

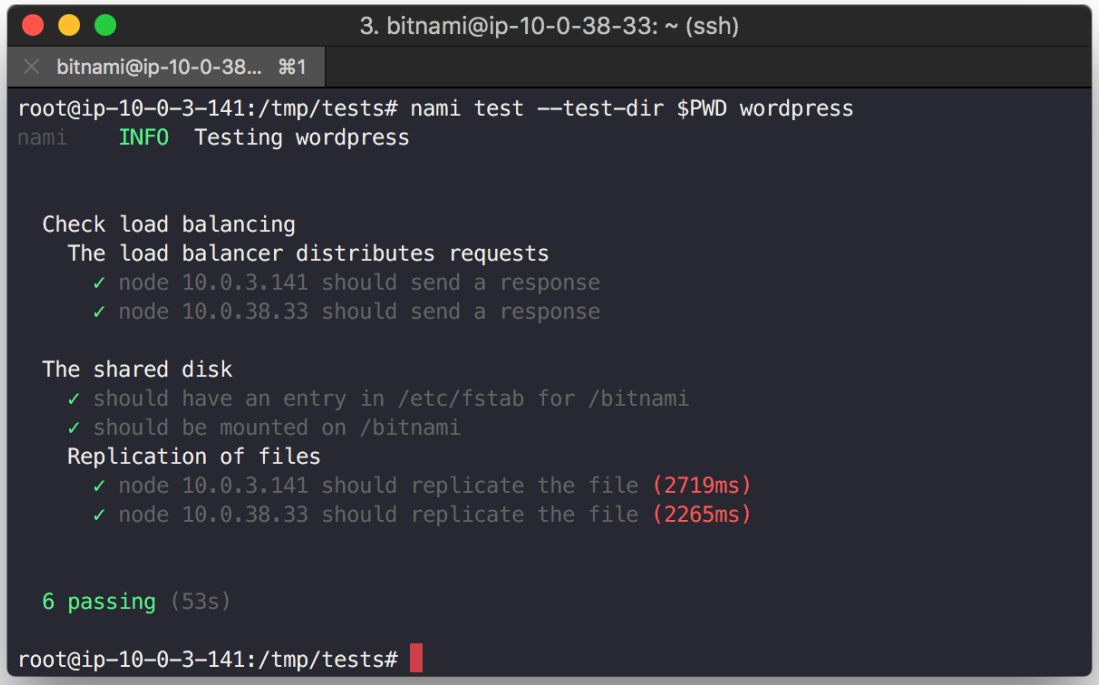


Figura 7-7. Salida tras la ejecución de los tests de verificación para el balanceador de carga y el sistema de ficheros compartido.

7.2.3 Tests de las plantillas

Por ultimo, aunque no se hace de forma automática mediante *Jenkins*[50], también se realizan chequeos con respecto a los parámetros recibidos por las plantillas *CloudFormation*. Para ello se utiliza la herramienta TaskCat (https://github.com/aws-quickstart/taskcat) que comprueba que la solución se puede desplegar correctamente en todas las regiones soportadas. Este chequeo es realizado por el equipo de *AWS Quick Starts*.

7.3 Publicación

Una vez que la solución es testeada, se puede pasar a la fase de publicación. Para ello, de nuevo se utiliza *BRadmin*. La publicación consta de dos pasos. Por un lado, se almacenan las plantillas en un *bucket S3* para poder compartirlas si fuese necesario. Por otra parte, se publica las *AMI* construida en *AWS*. En este proceso se copia la imagen en todas las regiones soportadas. El *Anexo F: Configuración de BRadmin* recoge los métodos más relevantes que han sido implementados para dar soporte a esta solución en el paso de publicación. En la *Figura 7-8* se esquematizan las acciones realizadas.

[50] El motivo de esto es que AWS ya realiza este tipo de pruebas por su parte.

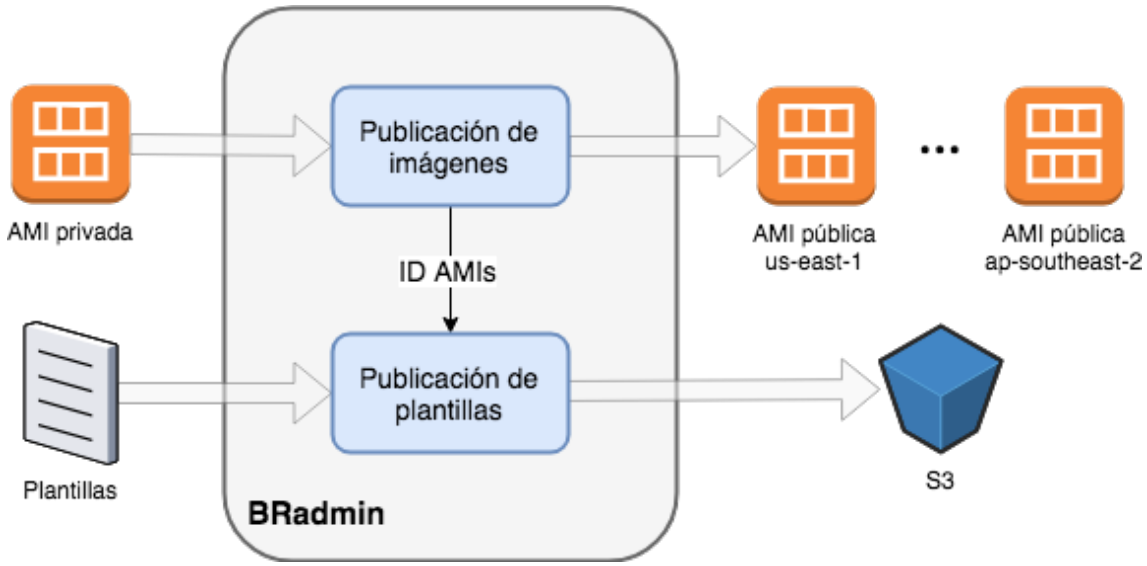


Figura 7-8. Proceso de publicación de AMIs y plantillas.

8 FASES DE DESARROLLO

A lo largo de este capítulo se pretende hacer un recorrido por las distintas fases de desarrollo por las que ha ido pasando el proyecto, haciendo hincapié en aquellas partes que han resultado más complejas o que tengan un especial interés para dar una visión global de la envergadura del proyecto completo.

8.1 Tecnologías y herramientas utilizadas

En la *Tabla 8-1* se muestra esquema que representa las fases por las que ha ido pasando el proyecto, así como las herramientas y tecnologías utilizadas en cada una de ellas. Como es lógico en el desarrollo de software y tecnología en general, se pasa por las fases más de una vez a lo largo del proyecto para aplicar cambios requeridos por el cliente o solucionar errores. Por ejemplo, la fase de diseño de las plantillas *CloudFormation* se ha ido repitiendo en múltiples ocasiones.

En las secciones posteriores, se desarrolla en mayor medida cada una de las fases.

Tabla 8-1. Fases de desarrollo del proyecto.

Fase	Objetivo	Lenguaje	Herramientas
Organización inicial del proyecto	Subdividir el proyecto en tareas.	-	Phabricator, arcanist
Implementación de la cloud AWS en Provisioner	Definir métodos de comunicación entre una cloud y Provisioner.	Typescript	Provisioner
Declaración de la definición de Provisioner	Especificar la información que describe la solución.	YAML	-
Construcción del provisioner-bundle	Crear un archivo con todos los artefactos necesarios para realizar la instalación y configuración de las instancias.	Typescript, Javascript	Provisioner, Docker, Gulp, Typescript compiler
Construcción de	Generar imágenes de cloud.	Ruby	BRadmin,

AMIs			aws-sdk, aws-cli
Diseño de las plantillas CloudFormation	Definir los recursos de cloud que están presentes en la solución	JSON	aws-cli, jq, TaskCat
Integración en repositorio Git	Incluir los ficheros en un repositorio de código	-	Git
Testeo de la solución	Definir compatibilidad de regiones y recursos y comprobación de despliegue de la solución	-	TaskCat
Integración de la solución en Jenkins	Incluir y configurar la solución en el sistema de integración continua.	Groovy, XML, Javascript	Jenkins
Integración de la solución en BRadmin	Permitir la realización de modificaciones en las plantillas CloudFormation para su posterior testeo	Ruby	BRadmin
Desarrollo de la guía de despliegue	Escribir la documentación que acompaña a la solución	-	Microsoft Word, Microsoft PowerPoint
Ampliación de la batería de tests	Crear nuevos tests para la solución	Javascript	CasperJS, Mocha, Chai, Bitnami Test System

8.2 Entorno de trabajo

Este *Trabajo Fin de Grado* se desarrolla como parte de un proyecto que *Bitnami* quiere llevar a cabo en colaboración con el equipo *AWS Quick Starts*. En *Bitnami*, el equipo que lo lidera es el de *Assets*[51], del que formo parte.

Las tareas principales del equipo son las de mantener y crear nuevos productos en distintas formas: instaladores, máquinas virtuales, imágenes de cloud o contenedores principalmente. Mediante acuerdos con otras compañías, se llevan a cabo proyectos puntuales de un carácter más personalizado. Ejemplos de esto son las soluciones clúster basadas en *Terraform* para *Oracle* (https://github.com/bitnami/oci-multi-tier) o *WordPress High Availability*.

Por otra parte, el equipo también se encarga de ofrecer soporte para ayudar a los usuarios de nuestros productos, principalmente a través del foro https://community.bitnami.com/.

No obstante, otros equipos también han colaborado en el proyecto. Por mencionar alguno, el equipo de documentación ha participado en la elaboración de la documentación técnica.

[51] El término Assets se refiere a productos de la compañía que tienen valor.

8.2.1 Organización

En este proyecto se han utilizado metodologías de trabajo ágiles [54] que ayudan a priorizar las tareas y organizar los equipos. Los proyectos se dividen en unidades de trabajo por tiempos no superiores a dos semanas. Este tiempo de dos semanas se denomina iteración. Las unidades de trabajo, se llaman *User Stories*.

Al principio de una iteración se realiza una estimación de la complejidad y tiempo que las tareas pueden conllevar. A diario, el equipo se reúne para comentar el progreso del día anterior y el plan del día. Al final de la iteración, se realiza una revisión del trabajo realizado y si las tareas se pueden considerar como completadas.

8.2.2 Software utilizado

Para la elaboración del proyecto se han utilizado diversas herramientas que se describen a continuación.

8.2.2.1 Visual Studio Code

https://code.visualstudio.com/

Se trata del editor de código utilizado durante todo el desarrollo del proyecto. Es desarrollado por Microsoft, gratuito y de software libre. Permite ampliar su funcionalidad básica de forma modular por medio del uso de plugins para el resaltado de sintaxis, integración con Git y una infinidad más de características.

8.2.2.2 Vim

https://github.com/vim/vim

Si bien se ha utilizado *Visual Studio Code* como principal editor de código, Vim ha sido utilizado a la hora de encontrar errores en la solución ya en funcionamiento, utilizado de forma remota por medio de conexiones *SSH*.

Entre los desarrolladores de software suele haber una gran polémica entre el uso de los editores *Vim* y *Emacs*. Personalmente, me resulta indiferente. La elección del uso de *Vim* ha sido por razones prácticas ya que *Vim* (o *Vi*) se encuentra preinstalado en todos los sistemas UNIX. De ahí que su uso sea conveniente en tareas de detección de errores, ya sea para observar el contenido de ficheros de logs o para realizar pequeñas modificaciones del código de las herramientas.

8.2.2.3 Git

https://git-scm.com/

Como herramienta para el control de versiones, se ha utilizado *Git*. Todos los repositorios de código de *Bitnami* se administran utilizando esta herramienta. Se trata de una herramienta de software libre diseñada por Linus Torvalds[52].

[52] Ingeniero de software también conocido por ser el desarrollador inicial del kernel de Linux.

Esta herramienta se integra con el repositorio de código *GitHub*, que se está utilizando para compartir el trabajo de forma externa con el equipo de *AWS Quick Starts*.

8.2.2.4 Phabricator y arcanist

https://github.com/phacility/phabricator

Phabricator es una aplicación de gestión de proyectos con herramientas orientadas al desarrollo de código. Entre sus principales funcionalidades, permite crear tareas y revisiones de código[53].

Phabricator se complementa con una herramienta por línea de comandos llamada *arcanist*. Sirve para crear revisiones de código, subir archivos o gestionar tareas desde la terminal.

8.2.2.5 Bitnami toolchain

Se trata de un conjunto de herramientas desarrolladas por *Bitnami*. Son numerosas, pero las que han sido de utilidad en este proyecto han sido principalmente cuatro.

- **Blacksmith**
  https://github.com/bitnami/blacksmith

  Para tareas de compilación. Recibe código fuente y devuelve código ejecutable.

- **Nami**
  https://github.com/bitnami/nami

  Herramienta para la configuración automática de aplicaciones. Su funcionamiento es explicado en más detalle en la sección 6.2.

- **Fisherman**
  Empaquetado de aplicaciones. Recibe los ficheros de la lógica de configuración (*Nami*) y los ficheros compilados (*Blacksmith*), creando un archivo denominado módulo de *Nami*.

- **Bitnami Test System**
  Herramienta para la ejecución de tests de aplicaciones en funcionamiento, bien sea en un entorno local o remoto.

8.2.2.6 aws-cli

https://github.com/aws/aws-cli

Es una herramienta por línea de comandos para la administración de los servicios de *AWS*. Se aportan más detalles en la sección 3.5.1.

[53] Una revisión de código impliaca compartir el código escrito con compañeros de equipo, normalmente en forma de diferencias respecto a una versión anterior. Permite detectar bugs o proponer mejoras.

8.2.2.7 Google Chrome y Development Tools

https://developers.google.com/web/tools/chrome-devtools

Se trata de un conjunto de herramientas que forman parte del navegador *Google Chrome* para el análisis de sitios web y diagnóstico de errores.

8.2.2.8 Jenkins

https://jenkins.io/

Jenkins es una aplicación de integración continua que permite realizar tareas automatizadas (como por ejemplo ejecutar tests de integración) ante la detección de cambios en el código de diferentes repositorios. Si bien esa es su funcionalidad principal, en *Bitnami*, *Jenkins* se utiliza como una interfaz para controlar y visualizar la construcción de artefactos, su despliegue y su testeo, de forma manual o automática.

8.2.2.9 Consola de BRadmin

Es una herramienta que forma parte del proyecto *BRadmin* (interno de *Bitnami*). Permite realizar tareas básicas como por ejemplo la creación de imágenes de *cloud* invocando métodos del código del proyecto. Resulta muy útil para detectar y resolver problemas.

8.2.2.10 Dockerdev

Dockerdev es un contenedor *Docker* orientado al desarrollo del día a día en *Bitnami*. Contiene las herramientas necesarias para poder trabajar en un entorno reproducible. Cada desarrollador de *Bitnami* tiene asignado uno de estos contenedores a los que se puede conectar mediante *SSH* puesto que los contenedores se lanzan en dos grandes instancias de *cloud*.

Esta forma de trabajar permite realizar tareas de alto coste computacional utilizando el 100% de los recursos de la instancia ya que no siempre todos los desarrolladores trabajan en paralelo. También su uso resulta muy conveniente para tareas que requieran la descarga de paquetes pesados puesto que el ancho de banda en la infraestructura *cloud* es muy superior al que se puede conseguir con la conexión de la oficina de Sevilla.

8.3 Fases de desarrollo

8.3.1 Solicitud del proyecto y primeros pasos

Normalmente para crear una nueva solución *AWS Quick Start* se debe seguir un procedimiento estándar (ver 3.4.2.1) y quien solicita el añadir la solución es quien la realiza. Sin embargo, el caso de este proyecto, es *AWS Quick Start* quien le solicita el proyecto a *Bitnami*.

En este punto el equipo de *Business Development*[54] de *Bitnami* es quien hace de intermediario con *AWS* y le dirige la solicitud a Ingeniería y se le asigna al equipo de *Assets*.

[54] Es el equipo que recibe y busca oportunidades de negocio.

El proyecto se desgrana en subtareas de forma que se puedan acometer por partes. En la *Figura 8-1* se muestra un ejemplo de la subdivisión de una *User Story*. En general, estas *User Stories* no hacen referencia a su implementación técnica o qué proyectos o partes de código hay que desarrollar.

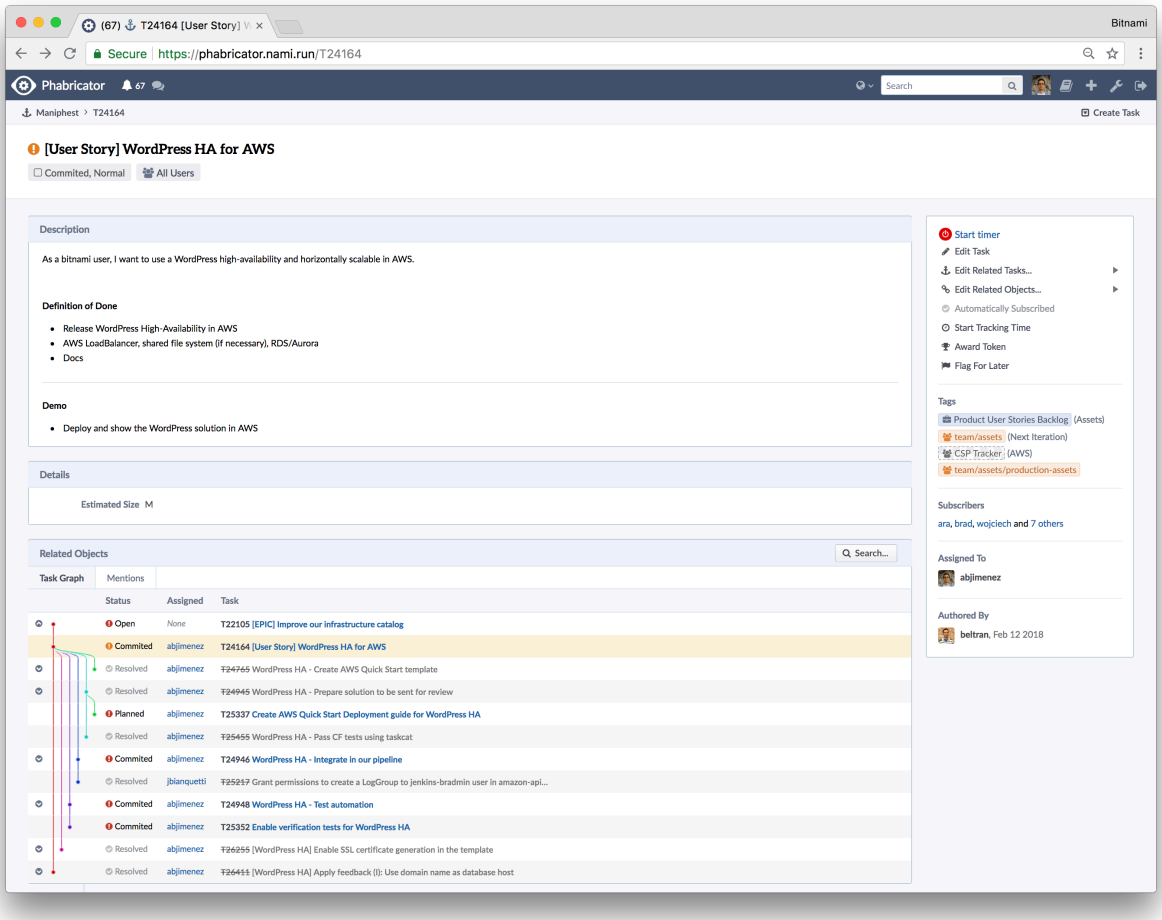


Figura 8-1. Vista en Phabricator de la User Story para WordPress High Availability.

8.3.2 Implementación de la cloud AWS en Provisioner

Una de las primeras tareas que hay que llevar a cabo es la adaptación de *Provisioner* a la *cloud AWS*. Su implementación es explicada en detalle en la sección 6.3.2 y se puede ver en el *Anexo A: Código fuente implementado de Provisioner*.

8.3.3 Construcción del provisioner-bundle

8.3.3.1 Definición de la información de la solución

Como se especifica en la sección 6.3.3 sobre la definición de *Provisioner*, es necesario especificar en algún punto la información sobre los parámetros que *Nami* recibe o los paquetes que hay que incluir en las imágenes de *cloud*.

8.3.3.2 Construcción del archivo

Una vez se tiene la definición y la lógica de provisioner, se puede construir el denominado *provisioner-bundle*. Esta construcción se realiza utilizando *Docker* y *Gulp* (https://gulpjs.com/), un gestor para la automatización de tareas de creación de artefactos de proyectos *JavaScript*.

Durante este proceso, también se realiza un análisis automático de las licencias de software de los paquetes que contiene el archivo, para su reporte a los proveedores *cloud*.

8.3.3.3 Resultado

El contenido del archivo resultante (llamado *provisioner-bundle*) es el mostrado en la *Tabla 8-2*. Se puede observar como contiene metadatos para la configuración de la máquina (*/root/.provisioner*), los módulos de *Nami* (*/opt/bitnami/**) y su lógica de configuración (*/root/.nami*).

Tabla 8-2. Contenido del provisioner-bundle.

```
├── opt
|   └── bitnami
|       ├── apache
|       ├── mysql
|       ├── php
|       ├── stats
|       └── wordpress
└── root
    ├── .nami
    |   ├── components
    |   └── registry.json
    └── .provisioner
        ├── configuration.json
        └── stackconfig.json
```

8.3.4 Primera versión de la solución

8.3.4.1 Construcción de AMIs

Teniendo preparado el *provisioner-bundle*, solo queda construir la imagen (*AMI*) tal y como se describe en la sección 6.1. Esta parte de construcción de imágenes la realiza un equipo distinto de *Bitnami*, *Webdev*. Este equipo se encarga, entre otras muchas cosas, del mantenimiento de la web de *Bitnami* (bitnami.com) o de definir interfaces para la conexión con las *APIs* de cada uno de los proveedores de servicios *cloud*.

8.3.4.2 Diseño de la primera versión de la solución

El siguiente paso en el proyecto es realizar un primer diseño de la plantilla de *CloudFormation* con una estructura bastante más primitiva que la de la versión final. La topología utilizada se puede ver en la *Figura 8-2*.

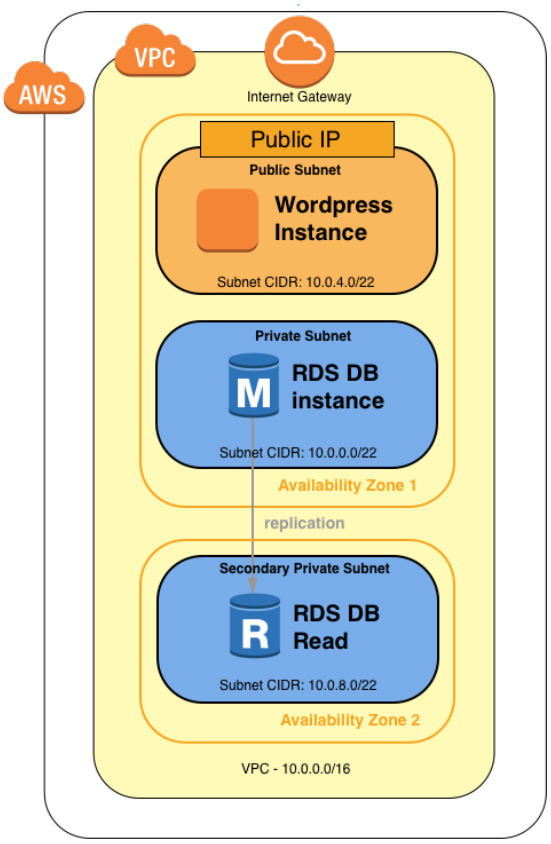


Figura 8-2. Diagrama de arquitectura de la primera versión de la solución.

Se puede observar como se trata de una versión de la solución mucho más simple. Llama la atención la presencia de un único nodo *frontend*. Por el hecho de funcionar con un único nodo, se elimina la necesidad de utilizar un balanceador de carga, el grupo de escalado automático y también el uso de un sistema de ficheros compartido.

La solución se ofrece en dos modalidades. Una plantilla que crea una nueva *VPC* y otra que despliega los recursos en una *VPC* ya existente. Las plantillas completas de esta versión primitiva se encuentran en el *Anexo C: Primera versión de las plantillas de CloudFormation*.

Esta arquitectura no cumple las exigencias mínimas en cuanto al concepto de alta disponibilidad. Si bien la base de datos gestionada por el servicio *RDS* sí se configura para su funcionamiento en múltiples zonas de disponibilidad, la instancia que ejecuta el servicio web solo esta disponible en una de las zonas. Si esta zona de disponibilidad sufriese alguna incidencia, el servicio dejaría de estar disponible.

En cuanto a seguridad, no se hace una separación clara entre subredes públicas y privadas y no se hace uso de las instancias *Bastión*. En esta arquitectura, la instancia frontend posee una dirección *IP* pública que es la que se utiliza para acceder al sitio web mediante *HTTP* y también la que se utiliza para acceder a la instancia de forma remota mediante *SSH*.

Aunque claramente no cumple con una amplia parte de los requisitos para las soluciones Quick Start (ver *Tabla 3-5 Requisitos técnicos de las soluciones Quick Start*), la solución es funcional y tras conversaciones con el equipo de *AWS Quick Start*, aceptan valorar si es posible añadirla como solución.

El proyecto queda en pausa a la espera de recibir una respuesta. Finalmente, se considera como no válida, principalmente porque no se adapta a las recomendaciones y mejores prácticas definidas por *AWS*.

8.3.5 Reimplementación de la solución

Puesto que la primera versión de la solución no es válida, se decide reimplementar la arquitectura utilizando las plantillas base que AWS proporciona para la definición de los recursos de red y para las instancias bastión. Además, se intenta jerarquizar todo lo posible el código de las plantillas por conjuntos de recursos.

Se va iterando sobre las plantillas añadiendo todos los recursos descritos en la sección 3.2 sobre recursos de *AWS*, y se van implementando las soluciones expuestas en el capítulo 4 sobre arquitectura de la aplicación.

8.3.5.1 Integración de los ficheros en el repositorio de AWS

Una vez que la solución cumple con la mayoría de requisitos, los ficheros se integran en el repositorio de código (https://github.com/aws-quickstart/quickstart-bitnami-wordpress) sin ser publicado. Llegados a este punto, las conversaciones con el equipo de *AWS* se agilizan, teniendo una revisión de código más concienzuda y aportando más ideas para la mejora de la solución. La comunicación suele ser vía correo electrónico o mediante videoconferencias.

La estructura final es la mostrada en la *Tabla 8-3*. Además de los directorios ya descritos (*submodules* y *templates*), se puede observar un tercer directorio, *ci* (integración continua). Este contiene una serie de ficheros con información sobre como desplegar la solución utilizando una herramienta denominada TaskCat (https://github.com/aws-quickstart/taskcat).

El contenido de este directorio incluye un fichero con valores de parámetros que dar a *CloudFormation* para la configuración de la solución y otros que permiten especificar en qué regiones se debe desplegar para su testeo. En la *Figura 8-3* se muestra el reporte que ofrece la herramienta tras su ejecución.

Tabla 8-3. Estructura final de las plantillas CloudFormation.

```
├── LICENSE.txt
├── NOTICE.txt
├── README.md
├── ci
|   ├── config.yml
|   ├── taskcat.yml
|   └── wordpress-master.json
├── submodules
|   ├── quickstart-aws-vpc
|   └── quickstart-linux-bastion
└── templates
    ├── rdsaurora.template
    ├── securitygroups.template
    ├── webserver.template
    ├── wordpress-master.template
    └── wordpress.template
```

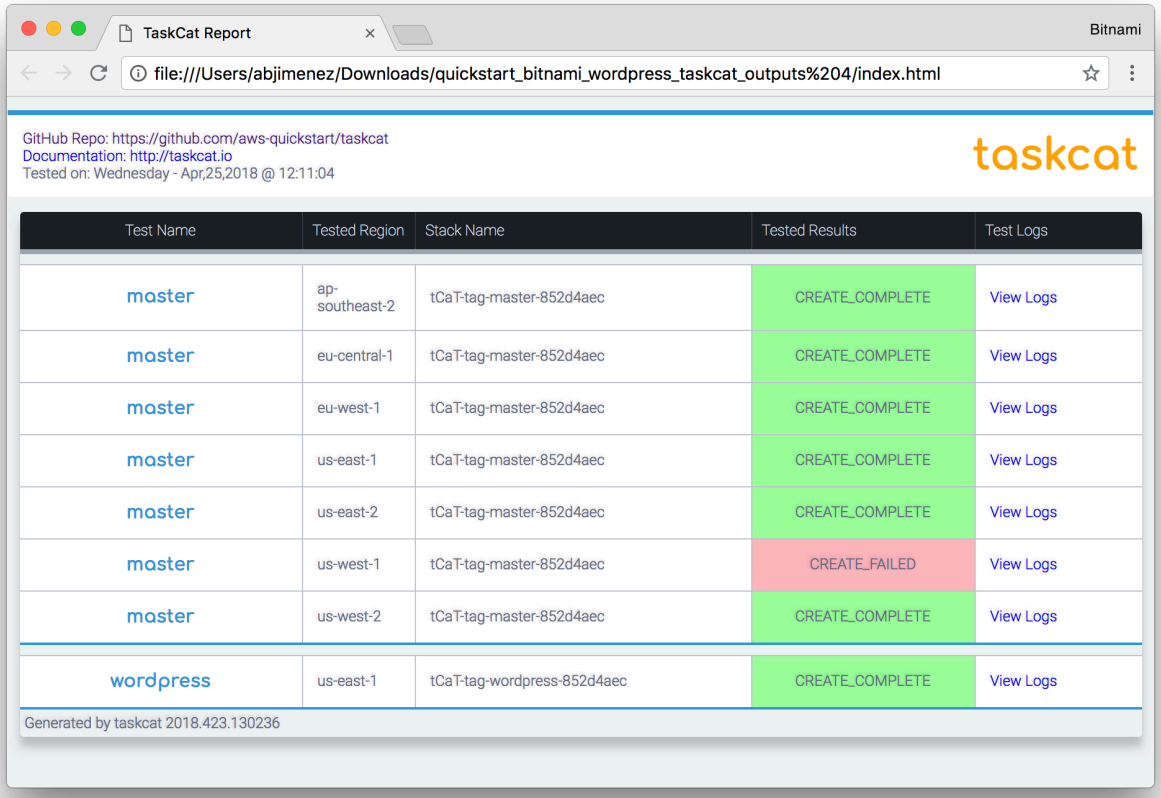


Figura 8-3. Reporte de la herramienta TaskCat.

Esta herramienta es ejecutada por parte de AWS cada vez que se solicita añadir nuevos cambios a las plantillas. Si el despliegue finaliza correctamente, los cambios propuestos son válidos para ser incluidos en la versión principal del código.

El contenido de todos los ficheros se puede ver en el *Anexo D: Plantillas de CloudFormation*.

8.3.5.2 Implementación de sugerencias

Las conversaciones con AWS se continúan y proponen que se integren nuevas funcionalidades como la generación de certificados SSL de forma automática (ver sección 5.5).

Aunque la solución no cumple con dos de los requisitos, el equipo de *AWS* lo acepta. El primero de los requisitos no cumplidos se refiere al soporte en todas las regiones de *AWS*. Esto es una característica deseable, pero el uso del recurso *Elastic File System* lo impide, puesto que es un tipo de servicio que no está disponible en todas las regiones [55].

El segundo de los requisitos es el de no tener una imagen predefinida que contenga el software, sino que se cree directamente con *CloudFormation* mediante scripts de instalación. Esta forma de funcionamiento no es muy conveniente puesto que *Bitnami* sigue unos procesos diferentes en cuanto a la construcción de imágenes. El equipo de *AWS* lo acepta.

También informan de pequeños fallos en la implementación, como por ejemplo el uso de la dirección *IP* de *RDS* para contactar a la base de datos. Lo correcto es utilizar el dominio, puesto que la *IP* puede cambiar. Se aplican esos pequeños cambios.

8.3.6 Nuevas funcionalidades requeridas en Provisioner

En paralelo a los puntos anteriores, por el hecho de incluir nuevos elementos de *cloud* como *Elastic File System*, *Provisioner* requiere de nuevas funcionalidades para poder desplegar las aplicaciones. Como se comenta en la sección 6.3, *Provisioner* soporta un mecanismo muy extensible de "recetas" mediante el cual se puede definir lógica de configuración que ejecutar en cierto momento del provisionado de la máquina. Las recetas principales que se han requerido se enumeran a continuación.

8.3.6.1 Señalización a CloudFormation.

Su función es la de enviar una señal a *CloudFormation* para confirmar el estado de la inicialización de la instancia por medio del uso de la herramienta *cfn-signal* (ver apartado 6.4.1.2). Se ejecuta tras la inicialización los módulos de nami. Puesto que su utilización requiere el uso de una herramienta, esta se instala mediante otra receta durante la construcción de la imagen.

8.3.6.2 Montado de sistema de ficheros compartido

- Argumentos: Recibe las opciones de montado y si se debe incluir una entrada en el fichero */etc/fstab.*
- Devuelve: El valor de esa variable.

El método monta un dispositivo en un determinado punto de montaje y si se desea, se puede añadir una línea en el fichero */etc/fstab* para que el dispositivo sea montado en futuros arranques del sistema. El método es llamado des de una receta de *Provisioner*. Para que el funcionamiento del sistema de ficheros compartido EFS, es necesaria la instalación del paquete *nfs-common*.

Es interesante mencionar el uso de las características de tipado que TypesScript soporta. Permite la definición de interfaces que definen como deben ser los argumentos que el método recibe. Puesto que las opciones de montaje pueden ser muy variadas, es conveniente estandarizarlas por medio del uso de interfaces.

La implementación se encuentra en el *Anexo A: Código fuente implementado de Provisioner*.

8.3.6.3 Gestión de inicialización utilizando disco compartido

Elastic File System es un sistema almacenamiento compartido y por ello, es necesario gestionar su acceso. Por una parte, la lectura y escritura concurrente de ficheros es gestionada por la implementación de *NFS*. Sin embargo, es necesario un tipo adicional de control.

Los módulos de *Nami* tienen un comportamiento diferente según si la aplicación está ya inicializada o no. Una vez el módulo es inicializado, se crea un fichero oculto */opt/bitnami/.initialized* que evita que se produzca una segunda inicialización si esta intentase ejecutar, por ejemplo, en una futura actualización de la aplicación.

Este sistema funciona correctamente si las inicializaciones se producen en intervalos de tiempo diferentes (de nuevo, por ejemplo, cuando una aplicación recibe una nueva versión y se desea actualizar) pero no contempla una situación en la que varios módulos de Nami se inicializan en el mismo intervalo de tiempo, como es el caso de los múltiples nodos *frontend* de la solución *WordPress High Availability*.

Estas dos situaciones se representan en la *Figura 8-4*. En la parte superior, con un único nodo, la segunda inicialización detecta el fichero *.initialized* y realiza una inicialización mínima utilizando los datos ya existentes creados en la primera inicialización. Sin embargo, en la situación con múltiples nodos inicializando al mismo tiempo, el nodo número 2 no detecta el fichero (puesto que la inicialización del 1 aun no ha terminado) y se pueden producir errores de solapamiento.

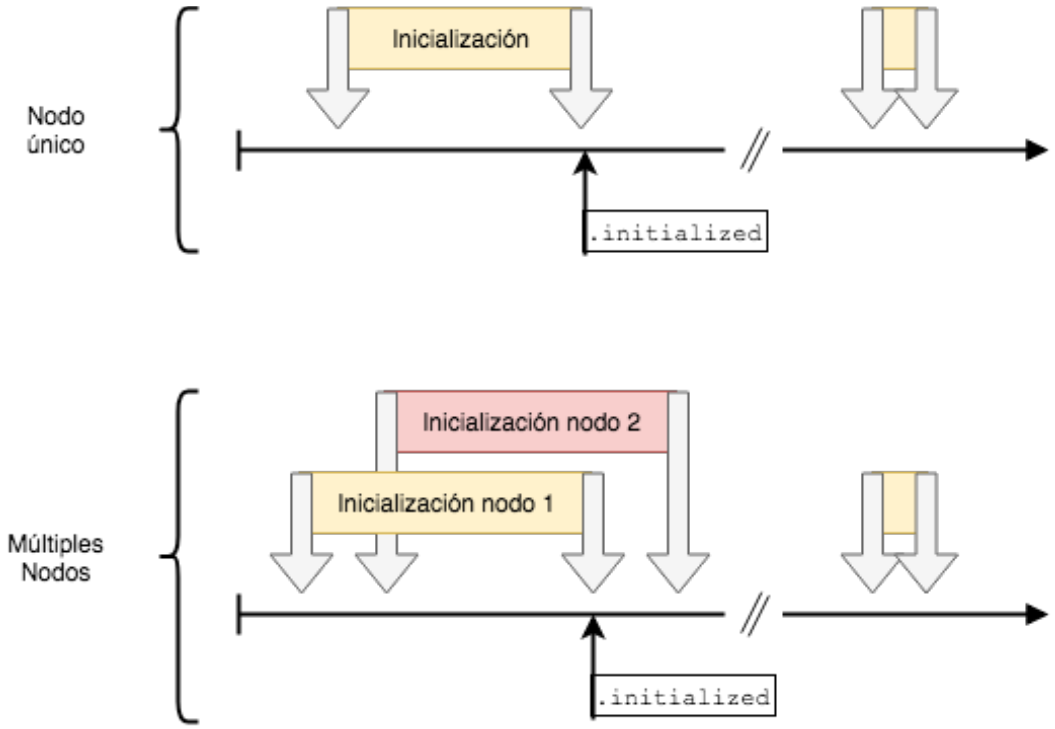


Figura 8-4. Diagramas temporales de inicialización de la aplicación.

Esta es la razón por la que se requiere un tipo adicional de control cuando existen recursos compartidos. Para solucionarlo, se ha implementado un método de inicialización bloqueante mediante una receta de *Provisioner*. El nodo que gane el acceso, es quien puede comenzar a inicializarse mientras que los demás esperan. Cuando un nodo libera el acceso, los demás pueden inicializar en paralelo, puesto que la inicialización en estos casos ya es mínima.

8.3.6.4 No almacenar credenciales en ficheros

Para el equipo de *AWS Quick Start* es necesario que las herramientas que gestionan la inicialización de las máquinas no almacenen credenciales en ficheros. Cuando se utiliza *user-data*, la información es almacenada en disco por parte de la utilidad *cloud-init*, en concreto en */var/lib/cloud/instance/user-data.txt* (*Figura 8-5*).

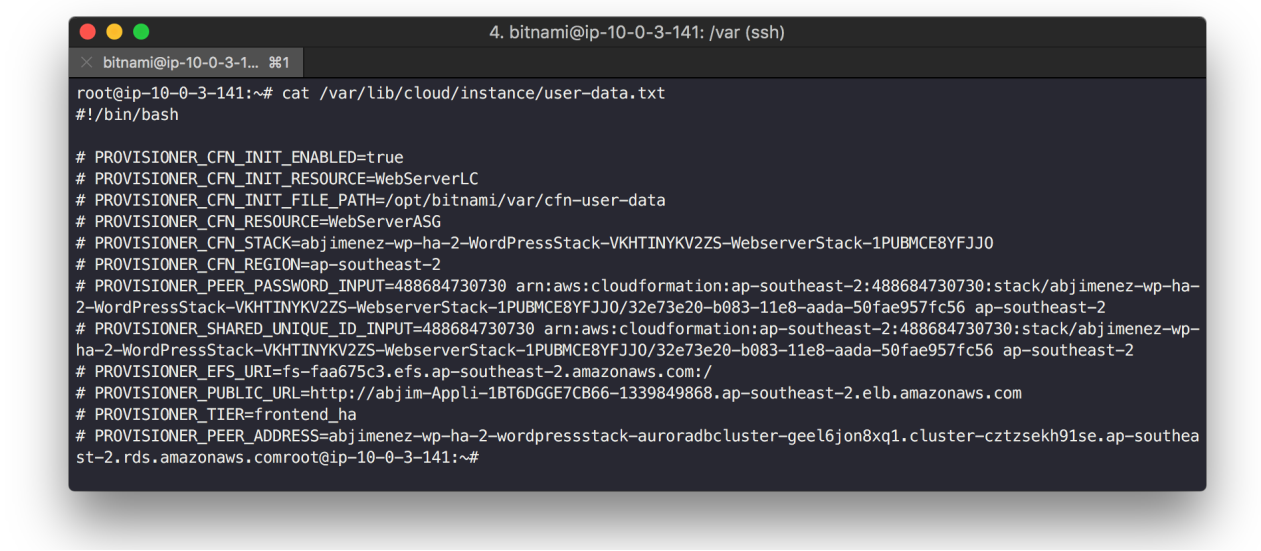


Figura 8-5. Contenido de user-data almacenado por cloud-init.

Por sea razón, las credenciales se gestionan utilizando *cfn-init* con la creación de un fichero que está bajo el control de *Provisioner*. Una vez que el fichero es utilizado, se elimina. Para ello, se han implementado los métodos *getExtraUserData* y *readExtraUserDataFile* que se pueden consultar en el *Anexo A: Código fuente implementado de Provisioner*.

8.3.7 Integración de la solución en los procesos de Bitnami

El siguiente paso es realizar los cambios necesarios en los procesos internos de *Bitnami* de forma que se puedan automatizar todas las tareas relativas al mantenimiento de la solución. Para ello, es necesario realizar modificaciones en la configuración de *Jenkins* e implementar nuevas funcionalidades en *BRadmin*, de forma que los pasos de construcción, despliegue, testeo y publicación de la solución queden totalmente automatizados.

El capítulo 7 explica con más detalle el proceso de integración.

8.3.8 Estimación de costes

Mediante la herramienta https://calculator.s3.amazonaws.com/index.html que *AWS* tiene, se procede a estimar los costes mensuales del despliegue de la solución. En la herramienta mencionada, se pueden ir añadiendo recursos y de esa manera poder calcular un presupuesto aproximado. El coste del despliegue de la solución *WordPress High Availability* ronda los 100-150 dólares mensuales.

8.3.9 Desarrollo de la guía de despliegue

Uno de los requisitos para poder publicar una solución en *AWS* y que esta se reconozca como *Quick Start* es que la solución debe ir acompañada de una guía que explique detalladamente cómo desplegar la aplicación, qué recursos se utilizan o información acerca de buenas prácticas. Esta parte ha sido un trabajo colaborativo entre *AWS* y *Bitnami* y dentro de Bitnami, el equipo de documentación ha colaborado de manera activa.

El resultado final de esta guía se puede ver en el siguiente enlace:

https://aws-quickstart.s3.amazonaws.com/quickstart-bitnami-wordpress/doc/wordpress-high-availability-by-bitnami-on-the-aws-cloud.pdf

8.3.10 Ampliación de la batería de tests para WordPress

Tal y como se explica en la sección 7.2 sobre el testeo de la aplicación, se implementan nuevos chequeos automatizados para esta solución, tanto tests funcionales como de verificación. Los tests ejecutados son realmente exigentes con la solución por lo que se puede tener garantía de que lo que se publica de forma automática funciona como se espera.

8.3.11 Lanzamiento y promoción de la solución

Como paso final, el equipo de *AWS Quick Start* junto con el equipo de Marketing de Bitnami se encargan de hacer pública la solución y de realizar acciones de promoción en redes sociales (ver *Figura 8-6*). A continuación, se muestran los diferentes productos que se hacen públicos:

- Sitio oficial de la solución: https://aws.amazon.com/quickstart/architecture/wordpress-high-availability-bitnami/
- Repositorio de código: https://github.com/aws-quickstart/quickstart-bitnami-wordpress
- Guía de despliegue: https://aws-quickstart.s3.amazonaws.com/quickstart-bitnami-wordpress/doc/wordpress-high-availability-by-bitnami-on-the-aws-cloud.pdf

Respecto a las acciones de promoción:

- Entrada en el blog "*What's new*" de *AWS*: https://aws.amazon.com/about-aws/whats-new/2018/09/deploy-wordpress-high-availability-with-bitnami-with-new-aws-quick-start/
- Redes sociales:
  - Facebook: https://www.facebook.com/1829098430460847
  - Twitter: https://twitter.com/awscloud/status/1041785279621853184
  - LinkedIn: https://www.linkedin.com/feed/update/urn:li:activity:6447550978135728128

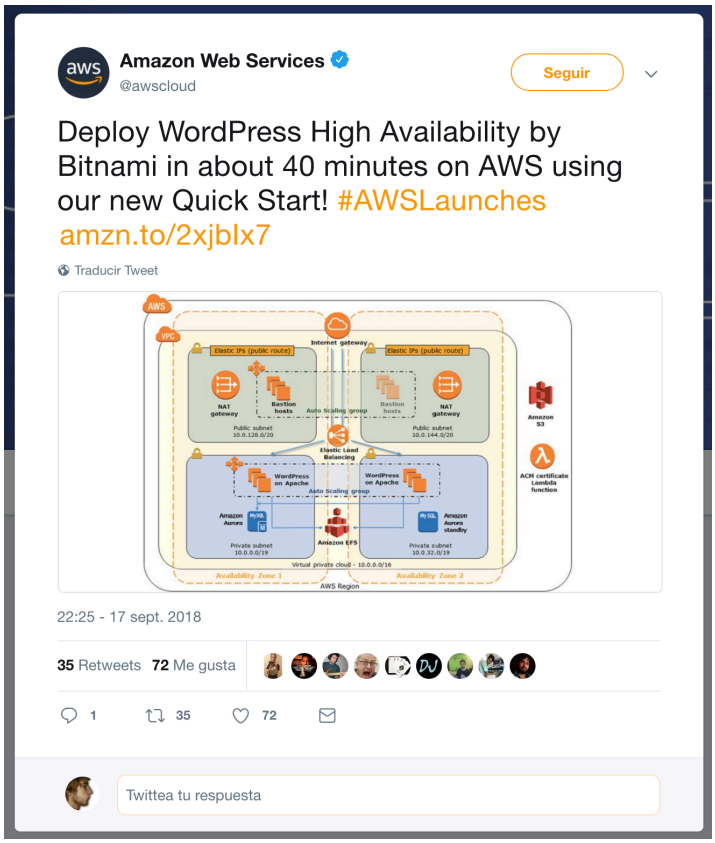


Figura 8-6. Promoción del lanzamiento de la solución en Twitter por parte de AWS.

9 CONCLUSIONES

Con la elaboración de esta memoria se pretende dar a conocer el trabajo que es necesario realizar para el desarrollo de soluciones para la *cloud* así como las implicaciones de migrar una aplicación tradicional como es WordPress a infraestructura *cloud* si se quiere aprovechar el máximo potencial de sus servicios.

Otro de los objetivos de este proyecto era el de realizar una estrecha colaboración con *AWS*, en concreto con el equipo de *Quick Start*. Fruto de esta colaboración *Bitnami* se presenta como socio colaborador de referencia para futuros acuerdos entre las dos organizaciones.

A nivel personal, este proyecto ha permitido aprender una infinidad de nuevas tecnologías dada la cantidad de sistemas, lenguajes de programación y herramientas implicadas. La realización de este trabajo abre las puertas para la realización de nuevos proyectos en entornos de *cloud*.

Por supuesto, tras el esfuerzo realizado, se espera que la solución tenga un éxito considerable. Si se quiere desplegar la aplicación *WordPress* con características de alta disponibilidad, esta es la solución de referencia más completa y avanzada que *Bitnami* y *AWS* ofrecen actualmente.

Esto no quiere decir que la solución descrita a lo largo de esta memoria no pueda continuar recibiendo mejoras. Parte del acuerdo con *AWS Quick Starts* consiste en realizar un mantenimiento y aportar mejoras que los usuarios soliciten y que sean de interés. Por mencionar algunas de estas mejoras, sería interesante continuar líneas de desarrollo enfocadas en la mejora del rendimiento de la solución.

Por un lado, por medio de redes de distribución de contenidos[55] (*CDN*) como *CloudFront* que permitan mejorar la experiencia de usuario disminuyendo los retardos de transmisión y conmutación al obtener los contenidos un servidor de destino más cercano. Gracias a los *CDN* también se consigue liberar de carga a los servidores principales y esto se traduce en un número menor de instancias en ejecución. Esto en parte, justifica el precio a pagar por replicar el contenido por todo el mundo.

[55] Una red de entrega de contenido o CDN (siglas del término anglosajón Content Delivery Network) está formada por un grupo de servidores proxy que almacenan copias de los datos servidos por un nodo principal de referencia a modo de caché. Los servidores se distribuyen geográficamente en diferentes localizaciones de forma que usuarios de distintas zonas del mundo pueden obtener el contenido de forma más eficiente.

A nivel de componentes de las *AMIs*, una de las mejoras que se podrían introducir es la utilización de *php-fpm*[56] en conjunción con *Apache*. De esta manera, el contenido estático de la aplicación *WordPress* es procesado y servido directamente por *Apache*, mientras que otro servicio adicional procesa las peticiones de contenido *PHP* dinámico. De esta forma, se obtiene un mejor rendimiento.

Por último, es importante mencionar que, si bien los servicios *cloud* siguen siendo la opción preferida para la modernización de aplicaciones y servicios de entornos empresariales, parece evidente que el futuro pasará por la utilización de contenedores. *Bitnami* está realizando una fuerte apuesta en este sentido, en concreto, mediante el desarrollo de soluciones de contenedores orquestadas por *Kubernetes*.

[56] PHP FastCGI Process Manager. Implementación de servicio PHP que se comunica con Apache por medio de la interfaz CGI para la interpretación de código PHP.

REFERENCIAS

[1] WordPress, «WordPress Mission,» [En línea]. Available: https://wordpress.org/about/.

[2] Amazon Web Services, Inc., «What is Cloud Computing?,» [En línea]. Available: https://aws.amazon.com/what-is-cloud-computing/.

[3] J. Barabas, «Defining IaaS, PaaS and SaaS,» [En línea]. Available: https://www.ibm.com/cloud/learn/iaas-paas-saas.

[4] E. Schouten, IBM® SmartCloud® Essentials, Packt Publishing Limited, 2013.

[5] Red Hat, Inc., «¿Qué es un contenedor de Linux?,» [En línea]. Available: https://www.redhat.com/es/topics/containers/whats-a-linux-container.

[6] C. Petters, «Build Your Own Face Recognition Service Using Amazon Rekognition,» 2017. [En línea]. Available: https://aws.amazon.com/blogs/machine-learning/build-your-own-face-recognition-service-using-amazon-rekognition/.

[7] Amazon Web Services, Inc., «Types of Cloud Computing,» [En línea]. Available: https://aws.amazon.com/types-of-cloud-computing/.

[8] Wikipedia, «Cloud computing,» [En línea]. Available: https://en.wikipedia.org/wiki/Cloud_computing.

[9] A. Regalado, «Who Coined 'Cloud Computing'?,» MIT Technology Review, 2011. [En línea]. Available: https://www.technologyreview.com/s/425970/who-coined-cloud-computing/.

[10] K. D. Foote, «A Brief History of Cloud Computing,» Dataversity, 2017. [En línea]. Available: http://www.dataversity.net/brief-history-cloud-computing/.

[11] J. v. Vliet y F. Paganelli, Programming Amazon EC2, Sebastopol: O'Reilly Media, 2011.

[12] J. Barr, «Amazon Simple Queue Service Released,» [En línea]. Available: https://aws.amazon.com/es/blogs/aws/amazon_simple_q/. [Último acceso: 2006].

[13] I. Pratt, «The Xen™ virtual machine monitor,» University of Cambridge Computer Laboratory, 2008. [En línea]. Available: https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/xen/.

[14] Wikipedia, «Timeline of Amazon Web Services,» [En línea]. Available: https://en.wikipedia.org/wiki/Timeline_of_Amazon_Web_Services.

[15] Amazon Web Services, Inc., «What is new (2017),» [En línea]. Available: https://aws.amazon.com/es/about-aws/whats-new/2017/.

[16] Amazon Web Services, Inc., «What is new (2018),» [En línea]. Available: https://aws.amazon.com/es/about-aws/whats-new/2018/.

[17] Amazon Web Services, Inc., «AWS Global Infrastructure,» [En línea]. Available: https://aws.amazon.com/about-aws/global-infrastructure/.

[18] Amazon Web Services, Inc., «Infraestructura de borde de Amazon CloudFront,» [En línea]. Available: https://aws.amazon.com/es/cloudfront/details/.

[19] RightScale, Inc., «RightScale 2018. State of the Cloud Report™,» 2018.

[20] Amazon Web Services, Inc., «Regions and Availability Zones,» [En línea]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[21] Amazon Web Services, Inc., «How AWS Pricing Works,» 2018. [En línea]. Available: https://d1.awsstatic.com/whitepapers/aws_pricing_overview.pdf.

[22] Amazon Web Services, Inc., «AWS GovCloud (US),» [En línea]. Available: https://aws.amazon.com/govcloud-us/.

[23] Amazon Web Services, Inc., «Amazon Elastic Compute Cloud: User Guide for Linux Instances,» 2018. [En línea]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf.

[24] Amazon Web Services, Inc., «Amazon EFS: How It Works,» [En línea]. Available: https://docs.aws.amazon.com/efs/latest/ug/how-it-works.html.

[25] Amazon Web Services, Inc., «What Is Amazon Relational Database Service (Amazon RDS)?,» [En línea]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html.

[26] Amazon Web Services, Inc., «Amazon Aurora FAQs,» [En línea]. Available: https://aws.amazon.com/rds/aurora/faqs/.

[27] Amazon Web Services, Inc., «Amazon Aurora Performance Assessment,» [En línea]. Available: https://d1.awsstatic.com/product-marketing/Aurora/RDS_Aurora_Performance_Assessment_Benchmarking_v1-2.pdf.

[28] Amazon Web Services, Inc., «How Elastic Load Balancing Works,» [En línea]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/how-elastic-load-balancing-works.html.

[29] Amazon Web Services, Inc., «AWS ElasticLoadBalancingV2 LoadBalancer,» [En línea]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-elasticloadbalancingv2-loadbalancer.html.

[30] Amazon Web Services, Inc., «Target Groups for Your Application Load Balancers,» [En línea]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html.

[31] Amazon Web Services, Inc., «Listeners for Your Application Load Balancers,» [En línea]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-listeners.html.

[32] Amazon Web Services, Inc., «What Is an Application Load Balancer?,» [En línea]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html.

[33] Amazon Web Services, Inc., «Auto Scaling Groups,» [En línea]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/AutoScalingGroup.html.

[34] Amazon Web Services, Inc., «AWS AutoScaling ScalingPolicy,» [En línea]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-policy.html.

[35] Amazon Web Services, Inc., «AWS CloudFormation User Guide,» [En línea]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-ug.pdf.

[36] Amazon Web Services, Inc., «AWS CloudFormation Features,» [En línea]. Available: https://aws.amazon.com/cloudformation/features/.

[37] Amazon Web Services, Inc., «Template Anatomy,» [En línea]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-anatomy.html.

[38] HashiCorp, «HCL is the HashiCorp configuration language,» [En línea]. Available: https://github.com/hashicorp/hcl.

[39] Amazon Web Services, Inc., «AWS Quick Starts,» [En línea]. Available: https://aws.amazon.com/quickstart/.

[40] Amazon Web Services, Inc., «Quick Start Contributor's Guide,» [En línea]. Available: https://aws-quickstart.github.io/.

[41] Amazon Web Services, Inc., «Tools for Amazon Web Services: SDKs,» [En línea]. Available: https://aws.amazon.com/tools/#sdk.

[42] Amazon Web Services, Inc., «AWS AutoScaling AutoScalingGroup,» [En línea]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-as-group.html.

[43] Amazon Web Services, Inc., «Modular and Scalable VPC Architecture,» [En línea]. Available: https://aws.amazon.com/quickstart/architecture/vpc/.

[44] Amazon Web Services, Inc., «Linux Bastion Hosts on AWS,» [En línea]. Available: https://aws.amazon.com/quickstart/architecture/linux-bastion/.

[45] Wikipedia, «Single point of failure,» [En línea]. Available: https://en.wikipedia.org/wiki/Single_point_of_failure.

[46] Amazon Web Services, Inc., «Creating an Amazon EBS-Backed Linux AMI,» [En línea]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/creating-an-ami-ebs.html.

[47] WordPress Foundation, «Installing WordPress: Famous 5-Minute Installation,» [En línea]. Available: https://codex.wordpress.org/Installing_WordPress#Famous_5-Minute_Installation.

[48] Amazon Web Services, Inc., «CloudFormation Helper Scripts Reference,» [En línea]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-helper-scripts-reference.html.

[49] Canonical Ltd, «cloud-init,» [En línea]. Available: https://cloud-init.io/.

[50] Bitnami, «L1 Terminal Fault: Privileged memory access vulnerability in Intel CPUs,» [En línea]. Available: https://blog.bitnami.com/2018/08/l1-terminal-fault-privileged-memory.html.

[51] Ruby-doc.org, «ERB - Ruby Templating,» [En línea]. Available: https://ruby-doc.org/stdlib-2.5.0/libdoc/erb/rdoc/ERB.html.

[52] Jenkins, «Jenkins User Documentation,» [En línea]. Available: https://jenkins.io/doc/.

[53] Apache Software Foundation, «Apache Module mod_headers,» [En línea]. Available: https://httpd.apache.org/docs/current/mod/mod_headers.html.

[54] Wikipedia, «Agile software development,» [En línea]. Available: https://en.wikipedia.org/wiki/Agile_software_development.

[55] Amazon Web Services, Inc., «Region Table,» [En línea]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/.

ANEXO A: CÓDIGO FUENTE IMPLEMENTADO DE PROVISIONER

- **A.1 cloud/aws.ts**
- **A.2 recipes/shared_disk.ts**
- **A.3 platform/linux.ts**

A.1 cloud/aws.ts

```
import { CloudOptions, CloudDataDiskDetails, CloudMetaData } from "./index";
import { BaseCloud } from "./base";
import { Cloud } from "./index";
import { file, os } from "../nami_wrapper";
import { Download } from "../download";
import * as fs from "fs";

import { gunzipSync } from "zlib";

/**
 * Logic specific to AWS cloud
 */
export class AwsCloud extends BaseCloud implements Cloud {
  // provided as protected so subclasses can change the value
  protected _apiUrl: string;
  private _userDataScriptOnce: Promise<string>;
  private _instanceIdentityDocumentOnce: Promise<any>;
  private _extraUserDataFileContents: string;

  constructor(options?: CloudOptions) {
    super(options);
    this._apiUrl = "http://169.254.169.254/latest";
    this.cloudTags = this.cloudTags.concat([
      "aws",
      "prebuilt-image",
      "hypervisor",
      "system-services"
    ]);
  }

  /**
   * For AWS, if PROVISIONER_CFN_INIT_ENABLED was passed in user data, call cfn-init
and try to read
   * extra user-data files.
```

```
   */
  async getUserData(key: string, now?: boolean): Promise<string> {
    if (now || !this._cloudUserData[key]) {
      let userdata = null;
      if (!this._extraUserDataFileContents) {
        await this._getExtraUserData();
      }
      if (this._extraUserDataFileContents && (this._extraUserDataFileContents !== ""))
{
        userdata = this._getParameterFromUserData(this._extraUserDataFileContents,
key);
      }
      this._cloudUserData[key] = userdata;
      if (this._cloudUserData[key]) {
        return this._cloudUserData[key];
      }
    }
    return super.getUserData(key, now);
  }

  /**
   * For AWS, download the user data script from metadata API
   */
  private _getUserDataScript(): Promise<string> {
    if (!this._userDataScriptOnce) {
      this._userDataScriptOnce = this._getUserDataScriptNow();
    }
    return this._userDataScriptOnce;
  }

  private async _getUserDataScriptNow(): Promise<string> {
    const dl: Download = new Download({
      returnData: true,
      returnBuffer: true,
      ignoreErrors: true
    });
    await dl.download(`${this._apiUrl}/user-data`);
    if (dl.statusCode < 400) {
      let body = dl.body;
      try {
        body = gunzipSync(dl.buffer).toString();
      } catch (e) {
        this.logger.debug("Unable to decompress user-data", e);
      }
      return body;
    } else {
      return undefined;
    }
  }

  /**
   * For AWS, get the data from user-data script
   */
```

```
  protected async _getUserDataNow(key: string): Promise<string> {
    const userDataScript = await this._getUserDataScript();
    if (!userDataScript) {
      return "";
    }
    return this._getParameterFromUserData(userDataScript, key);
  }

  /**
   * Read extra user-data passed from CFN init files. The file is deleted after saving
the content
   */
  private readExtraUserDataFile(extraUserDataFilePath): void {
    this.logger.info(`Reading CFN user-data file ${extraUserDataFilePath}`);
    this._extraUserDataFileContents = file.read(extraUserDataFilePath);
    try {
      this.logger.debug(`Deleting CFN user-data file ${extraUserDataFilePath}`);
      file.delete(extraUserDataFilePath);
    } catch (e) {
      throw new Error(`Unable to delete CFN user-data file: ${e.message}`);
    }
  }

  /**
   * If PROVISIONER_CFN_INIT_ENABLED was passed in user data, call cfn-init and try to
read
   * extra user-data files.
   */
  private async _getExtraUserData(): Promise<void> {
    if (await this._getUserDataNow("PROVISIONER_CFN_INIT_ENABLED")) {
      try {
        os.runProgram("cfn-init", ["--verbose",
          "--stack", await this._getUserDataNow("PROVISIONER_CFN_STACK"),
          "--resource", await this._getUserDataNow("PROVISIONER_CFN_INIT_RESOURCE"),
          "--region", await this._getUserDataNow("PROVISIONER_CFN_REGION")
        ]);
      } catch (e) {
        throw new Error(`Unable to run cfn-init: ${e.message}`);
      }
      // if cfn-init created a user data file, read it
      const cfnUserDataFilePath: string = await
this._getUserDataNow("PROVISIONER_CFN_INIT_FILE_PATH");
      if (file.exists(cfnUserDataFilePath)) {
        this.readExtraUserDataFile(cfnUserDataFilePath);
      }
    }
  }

  /**
   * Get instance identity document and parse it as JSON
   */
  protected _getInstanceIdentityDocument(): Promise<any> {
    if (!this._instanceIdentityDocumentOnce) {
```

```
      this._instanceIdentityDocumentOnce = this._getInstanceIdentityDocumentNow();
    }
    return this._instanceIdentityDocumentOnce;
  }

  protected async _getInstanceIdentityDocumentNow(): Promise<any> {
    const dl: Download = new Download({
      returnData: true,
      ignoreErrors: true
    });
    await dl.download(`${this._apiUrl}/dynamic/instance-identity/document`);
    return JSON.parse(dl.body);
  }

  protected async _getMetaDataNow(key: CloudMetaData): Promise<string> {
    if (key === "account-id") {
      const data: any = await this._getInstanceIdentityDocument();
      return data.accountId || "";
    } else {
      return super._getMetaDataNow(key);
    }
  }
};
```

A.2 recipes/shared_disk.ts

```
/// <reference path="../../typings-recipe.d.ts" />

"use strict";
(() => {
  /**
   * Install packages for EFS support (Elastic File System)
   * https://docs.aws.amazon.com/efs/latest/ug/mounting-fs.html#mounting-fs-install-nfsclient
   */
  recipes.register({
    id: "shared-disk-packages-aws",
    on: {provisionMachine: {depends: ["system-packages"]}},
    conditions: {
      tierTags: {any: ["requires-shared-disk"]},
      platformTags: {any: ["debian"]},
      cloudTags: {any: ["aws"]}
    },
    recipeHandler: async function(input) {
      await utils.retry(() => {
        platform.installPackages("nfs-common");
      });
    }
  });

  /**
   * Mount shared file system
   */
  recipes.register({
    id: "shared-disk-aws",
    on: {beforeInitialize: {}},
    conditions: {
      instanceTier: {not: ["main"]},
      tierTags: {any: ["requires-shared-disk"]},
      cloudTags: {any: ["aws"]}
    },
    recipeHandler: async function(input) {
      const efsUri: string = await cloud.getUserData("PROVISIONER_EFS_URI");
      const mountConfig: any = {
        device: efsUri,
        mountPoint: platform.pathInfo.namiDataPath,
        type: "nfs4",
        options: {
          nfsvers: "4.1",
          rsize: 1048576,
          wsize: 1048576,
          hard: true,
        }
      };
      platform.mount(mountConfig, {addToFstab: true});
    }
  });
```

```
  const initializingFile: string = $file.join(platform.pathInfo.namiDataPath,
".initializing");
  const initializedFile: string = $file.join(platform.pathInfo.namiDataPath,
".initialized");

  /**
   * Try to lock the shared disk to perform the first initialization.
   */
  recipes.register({
    id: "shared-disk-lock-initialization",
    on: {beforeInitialize: {depends: ["shared-disk-aws", "shared-disk-azure"]}},
    conditions: {
      tierTags: {any: ["requires-shared-disk"]},
      shouldInvoke: () => { return !$file.exists(initializedFile); },
    },
    recipeHandler: async function(input) {
      const waitUntilInitialized = async function(): Promise<void> {
        await utils.retry(
          () => {
            if (!$file.exists(initializedFile)) {
              throw new Error(`Timeout: ${initializedFile} does not exist`);
            }
          },
          {
            attempts: 120, // 10 minutes
            delay: 5, // 5 seconds
          }
        );
      };

      await utils.delayMs(Math.random() * 30000); // [0,30] seconds
      if ($file.exists(initializingFile)) {
        logger.debug("The initializing lock file already exists");
        await waitUntilInitialized();
      } else {
        logger.debug("The initializing lock file does not exist yet. Trying to lock
for initialization");
        const contentCheck: string = Math.random().toString();
        $file.write(initializingFile, contentCheck);
        await utils.delayMs(3000); // Wait 3 seconds; assuming 3 > max sync time in
the shared disk
        if ($file.read(initializingFile) !== contentCheck) { // Avoid race conditions
          logger.debug("Could not lock for initialization.");
          await waitUntilInitialized();
        }
      }
    }
  });

  /**
   * Free the shared disk and mark it as initialized. The order in the initialization
is already granted.
```

```
   */
  recipes.register({
    id: "shared-disk-free-initialization",
    on: {afterInitialize: {}, afterFailedInitialize: {}},
    conditions: {
      tierTags: {any: ["requires-shared-disk"]},
      shouldInvoke: () => { return !$file.exists(initializedFile); },
    },
    recipeHandler: async function(input) {
      if (input.eventName === "afterInitialize") {
        $file.touch(initializedFile);
      }
      $file.delete(initializingFile);
    }
  });
})();
```

A.3 platform/linux.ts

```
import * as _ from "lodash";
import { Platform } from "./index";
import { BasePlatform } from "./base";
import { file, os } from "../nami_wrapper";

/**
 * Definition of mount options
 */
export interface MountConfig {
  /**
   *  Device to mount
   */
  device: string;
  /**
   * Directory where the device is going to be mounted
   */
  mountPoint: string;
  /**
   * Filesystem type
   */
  type: string;
  /**
   * Options object
   */
  options?: MountOptions;
}

/**
 * Definition of mount options for a network device
 */
export interface MountConfigShared extends MountConfig {
  options: MountOptionsShared;
}

export interface MountOptions {
  nofail?: boolean; // Do not report errors for this device if it does not exist
}

export interface MountOptionsShared extends MountOptions {
  vers?: string;          // SMB version
  nfsvers?: string;       // NFS protocol version
  username?: string;      // user to connect as
  password?: string;      // specifies the password
  file_mode?: string;     // overrides the default file mode
  dir_mode?: string;      // overrides the default mode for directories
  uid?: string | number;  // sets the uid that will own all files or directories
  gid?: string | number;  // sets the gid that will own all files or directories
  serverino?: boolean;    // Use inode numbers returned by the server
  guest?: boolean;        // don't prompt for a password
  rsize?: number;         // Data block size in bytes, to be transferred for reads.
Defauts to 32768 for NFSv4
```

```
  wsize?: number;         // Data block size in bytes, to be transferred for writes.
Defauts to 32768 for NFSv4
  hard?: boolean;         // The program using a file should stop and wait for NFS
connection
}


/**
 * Class for all Linux based operating systems
 */
export abstract class LinuxPlatform extends BasePlatform implements Platform {

  /**
  * Mount a device in a given directory. If specified it adds a new entry in
  * the `/etc/fstab` file.
  * @param {MountOptions|MountOptionsShared} mountConfig Options for the mount command
  * @param {Object} [extraOptions]
  * @param {boolean} [extraOptions.addToFstab] Add entry in fstab to mount in every
boot
  */
  mount(mountConfig: MountConfig | MountConfigShared, extraOptions?: {addToFstab?:
boolean}): void {
    // set default options
    const options = Object.assign({
      addToFstab: false
    }, typeof extraOptions === "undefined" ? {} : extraOptions);

    // Convert options object to a comma-separated string
    const mountOptionsString = (options): string => {
      let optionsString: string = "";
      for (const key in options) {
        let opt: string;
        if (typeof options[key] === "boolean") {
          if (options[key] === true) {
            opt = key;
          } else {
            continue;
          }
        } else {
          opt = `${key}=${options[key]}`;
        }
        optionsString = optionsString.concat(`${opt},`);
      }
      optionsString = optionsString.slice(0, -1);
      return optionsString;
    };

    // create mountpoint if not created
    file.mkdir(mountConfig.mountPoint);

    this.logger.debug("Mount configuration", mountConfig);
    os.runProgram("mount", [
      mountConfig.device,
```

```
      mountConfig.mountPoint,
      "--type", mountConfig.type,
      "--options", mountOptionsString(mountConfig.options)
    ]);

    if (options.addToFstab) {
      mountConfig.options.nofail = true; // Avoid VM hang during boot
      const fstabFile = "/etc/fstab";
      if (file.contains(fstabFile, mountConfig.device) || file.contains(fstabFile,
mountConfig.mountPoint)) {
        const fstabEntryRegExp = new
RegExp(`\\s?${mountConfig.device}\\s${mountConfig.mountPoint}.*`);
        if (file.contains(fstabFile, fstabEntryRegExp)) {
          // fstab contains an entry with the device and the mount point
          this.logger.info(`Updating ${mountConfig.device} entry in fstab`);
          file.substitute(fstabFile,
            fstabEntryRegExp,
            `${mountConfig.device} ${mountConfig.mountPoint} ` +
              `${mountConfig.type} ${mountOptionsString(mountConfig.options)}`
          );
        } else {
          // fstab contains an entry with the device or mount point but not for the
given mount point
          throw new Error(`${mountConfig.device}/${mountConfig.mountPoint} entries in
${fstabFile} are not compatible`);
        }
      } else {
        this.logger.info(`Adding ${mountConfig.device} to ${fstabFile}`);
        file.append(fstabFile, `${mountConfig.device} ${mountConfig.mountPoint} ` +
          `${mountConfig.type} ${mountOptionsString(mountConfig.options)}`);
      }
    }
  }
}
```

ANEXO B: CONTENIDO DE LA DEFINICIÓN DE PROVISIONER

B.1 definitions/wordpress.yml

```
---
details:
  key: wordpress
  name: WordPress
  mainModule: wordpress
supportedPlatforms:
  - distro: debian-9
    arch: amd64
    os: linux
    bitnamiPlatforms: []
defaults:
  username: user
  password: bitnami
dependencies:
  systemPackages: []
tiers:
  main:
    tags:
      - requires-shared-disk
    ports:
      public: [80, 443]
      internal: [3306]
    peerHostnames:
      local:
        - database
        - mariadb
        - mysql
    modules:
      - name: mariadb
        branch: '10.2'
        startAfterInitialize: true
        initializeArguments:
          - '--rootPassword'
          - value: peerPassword
      - name: php
        skipService: true
      - name: apache
      - name: libphp
      - name: mysql-client
        branch: '10.2'
        initializeArguments:
```

```
        - '--rootPassword'
        - value: peerPassword
        - '--createDatabaseName'
        - 'bitnami_wordpress'
        - '--createDatabaseUser'
        - 'bn_wordpress'
        - '--createDatabasePassword'
        - 'notARandomPassword'
    - name: wordpress
      initializeArguments:
        - '--databaseName'
        - 'bitnami_wordpress'
        - '--databaseUser'
        - 'bn_wordpress'
        - '--databasePassword'
        - 'notARandomPassword'
        - '--username'
        - value: appUsername
        - '--password'
        - value: appPassword
frontend:
  peerHostnames:
    default:
      - database
      - mariadb
      - mysql
  ports:
    public: [80, 443]
  waitForPorts:
    - 3306
  modules:
    - name: php
      skipService: true
    - name: apache
    - name: libphp
    - name: mysql-client
      branch: '10.2'
      initializeArguments:
        - '--host'
        - value: peerAddressHostname
        - '--rootPassword'
        - value: peerPassword
        - '--createDatabaseName'
        - 'bitnami_wordpress'
        - '--createDatabaseUser'
        - 'bn_wordpress'
        - '--createDatabasePassword'
        - generatePassword: wordpress
    - name: wordpress
      initializeArguments:
        - '--databaseServerHost'
        - value: peerAddressHostname
        - '--databaseName'
```

```
          - 'bitnami_wordpress'
          - '--databaseUser'
          - 'bn_wordpress'
          - '--databasePassword'
          - generatePassword: wordpress
          - '--username'
          - value: appUsername
          - '--password'
          - value: appPassword
  frontend_ha:
    tags:
      - requires-shared-disk
      - fixed-public-url
    default:
      - database
      - aurora
    ports:
      internal: [80]
    waitForPorts:
      - 3306
    modules:
      - name: php
        skipService: true
      - name: apache
      - name: libphp
      - name: mysql-client
        branch: '10.2'
        initializeArguments:
          - '--host'
          - value: peerAddress
          - '--rootPassword'
          - value: peerPassword
          - '--createDatabaseName'
          - 'bitnami_wordpress'
          - '--createDatabaseUser'
          - 'bn_wordpress'
          - '--createDatabasePassword'
          - generatePassword: wordpress
      - name: wordpress
        initializeArguments:
          - '--databaseServerHost'
          - value: peerAddress
          - '--databaseName'
          - 'bitnami_wordpress'
          - '--databaseUser'
          - 'bn_wordpress'
          - '--databasePassword'
          - generatePassword: wordpress
          - '--username'
          - value: appUsername
          - '--password'
          - value: appPassword
  database:
```

```
    ports:
      internal: [3306]
    modules:
      - name: mariadb
        branch: '10.2'
        initializeArguments:
          - '--rootPassword'
          - value: peerPassword
```

ANEXO C: PRIMERA VERSIÓN DE LAS PLANTILLAS DE CLOUDFORMATION

C.1 templates/wordpress.template

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Deploy WordPress on AWS creating a new VPC",
  "Metadata": {
      "AWS::CloudFormation::Interface": {
      "ParameterGroups": [
        {
          "Label": {
            "default": "Application Configuration"
          },
          "Parameters": [
            "ApplicationUsername",
            "ApplicationPassword",
            "InstancesType",
            "InstancesDataDiskType",
            "InstancesDataDiskSize"
          ]
        },
        {
          "Label": {
            "default": "Networking Configuration"
          },
          "Parameters": [
            "SSHKeyName",
            "IPRangeSSH",
            "IPRangeApplication",
            "AvailabilityZone",
            "SecondAvailabilityZone"
          ]
        },
        {
          "Label": {
            "default": "Database Configuration"
          },
          "Parameters": [
            "DatabasePassword",
            "DatabaseInstanceClass",
            "DatabaseAllocatedStorage"
          ]
        }
      ],
```

```
      "ParameterLabels": {
        "ApplicationUsername": {
          "default": "WordPress Admin Username"
        },
        "ApplicationPassword": {
          "default": "WordPress Password"
        },
        "InstancesType": {
          "default": "Application Instance type"
        },
        "InstancesDataDiskType": {
          "default": "Data disk type"
        },
        "InstancesDataDiskSize": {
          "default": "Data disk size"
        },
        "SSHKeyName": {
          "default": "SSH Key Name"
        },
        "IPRangeSSH": {
          "default": "SSH Source"
        },
        "IPRangeApplication": {
          "default": "Application Source"
        },
        "AvailabilityZone": {
          "default": "Availability zone"
        },
        "SecondAvailabilityZone": {
          "default": "Secondary zone"
        },
        "DatabaseInstanceClass": {
          "default": "Database class"
        },
        "DatabasePassword": {
          "default": "MariaDB Admin Password"
        },
        "DatabaseAllocatedStorage": {
          "default": "Allocated storage"
        }
      }
    }
  },
  "Parameters": {
    "InstancesDataDiskType": {
      "Description": "Disk type",
      "Type": "String",
      "Default": "gp2",
      "AllowedValues": [
        "standard",
        "gp2"
      ]
    },
```

```
    "InstancesDataDiskSize": {
      "Description": "Disk size in GB, This must be between 1 and 1024 GB",
      "Type": "Number",
      "MinValue": "1",
      "MaxValue": "1024",
      "Default": "10",
      "ConstraintDescription": "Your Data Disk Instance disk size must be between 1
and 1024 GB"
    },
    "ApplicationPassword": {
      "Description": "Admin Password for WordPress - Between 8-32 characters, include
at least one digit, upper case letter and no special characters",
      "Type": "String",
      "MinLength": 8,
      "MaxLength": 32,
      "AllowedPattern": "(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])^([a-zA-Z0-9]{8,32})",
      "ConstraintDescription": "Your password must be between 8 and 32 characters,
include at least one digit, upper case letter and no special characters",
      "NoEcho": true
    },
    "ApplicationUsername": {
      "Description": "IMPORTANT: This is the admin username you will use to log into
the application, and the password you set below will correspond to this admin
username. This is a default value, and can only be changed after you log in",
      "Type": "String",
      "Default": "user",
      "AllowedValues": [
        "user"
      ]
    },
    "SSHKeyName": {
      "Description": "SSH Key Name.",
      "Type": "AWS::EC2::KeyPair::KeyName",
      "AllowedPattern": ".+",
      "ConstraintDescription": "Your SSH Key cannot be blank. Please add an SSH key to
the application"
    },
    "IPRangeSSH": {
      "Description": "Set the IP or IP range for SSH connectivity to machine;
127.0.0.1/32 is configured by default to disable remote SSH access. Set this to
0.0.0.0/0 to access secure shell from any IP",
      "Type": "String",
      "MinLength": "9",
      "MaxLength": "18",
      "Default": "127.0.0.1/32",
      "AllowedPattern":
"(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
      "ConstraintDescription": "Your IP Range for SSH must be a valid IP CIDR range of
the form x.x.x.x/x"
    },
    "IPRangeApplication": {
      "Description": "This value defaults to empty. Set this to 0.0.0.0/0 to allow
anyone on the internet to access the application, and 127.0.0.1/32 to disable public
```

```
access to the application. You can also configure specific IP address or IP Range
access here",
      "Type": "String",
      "MinLength": "9",
      "MaxLength": "18",
      "AllowedPattern":
"(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})\\.(\\d{1,3})/(\\d{1,2})",
      "ConstraintDescription": "This value cannot be empty. Set this to 0.0.0.0/0 to
allow anyone on the internet to access the application, and 127.0.0.1/32 to disable
public access to the application"
    },
    "AvailabilityZone": {
      "Type": "AWS::EC2::AvailabilityZone::Name",
      "Description": "Primary availability zone for your deployment"
    },
    "SecondAvailabilityZone": {
      "Type": "AWS::EC2::AvailabilityZone::Name",
      "Description": "Second availability zone for RDS"
    },
    "InstancesType": {
      "Description": "Select the instance type for the WordPress instance",
      "Type": "String",
      "Default": "t2.medium",
      "AllowedValues": [
        "t2.small",
        "t2.medium",
        "t2.large",
        "t2.xlarge",
        "t2.2xlarge",
        "m3.medium",
        "m3.large",
        "m3.xlarge",
        "m3.2xlarge",
        "m4.large",
        "m4.xlarge",
        "m4.2xlarge",
        "m4.4xlarge",
        "m4.10xlarge",
        "m4.16xlarge",
        "c3.large",
        "c3.xlarge",
        "c3.2xlarge",
        "c3.4xlarge",
        "c3.8xlarge",
        "c4.large",
        "c4.xlarge",
        "c4.2xlarge",
        "c4.4xlarge",
        "c4.8xlarge",
        "r3.large",
        "r3.xlarge",
        "r3.2xlarge",
        "r3.4xlarge",
```

```
        "r3.8xlarge",
        "r4.large",
        "r4.xlarge",
        "r4.2xlarge",
        "r4.4xlarge",
        "r4.8xlarge",
        "r4.16xlarge",
        "i2.xlarge",
        "i2.2xlarge",
        "i2.4xlarge",
        "i2.8xlarge"
      ]
    },
    "DatabasePassword": {
      "Description": "MariaDB Database Admin Password for WordPress - Between 8-32
characters, include at least one digit, upper case letter and no special characters",
      "Type": "String",
      "MinLength": 8,
      "MaxLength": 32,
      "AllowedPattern": "(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])^([a-zA-Z0-9]{8,20})",
      "ConstraintDescription": "Your Database Admin Password must be between 8 and 32
characters, include at least one digit, upper case letter and no special characters",
      "NoEcho": true
    },
    "DatabaseInstanceClass": {
      "Description": "Select the instance type for your application database",
      "Type": "String",
      "Default": "db.t2.medium",
      "AllowedValues": [
        "db.t2.small",
        "db.t2.medium",
        "db.t2.large",
        "db.m4.large",
        "db.m4.xlarge",
        "db.m4.2xlarge",
        "db.m4.4xlarge",
        "db.m4.10xlarge",
        "db.r3.large",
        "db.r3.xlarge",
        "db.r3.2xlarge",
        "db.r3.4xlarge",
        "db.r3.8xlarge",
        "db.m3.medium",
        "db.m3.large",
        "db.m3.xlarge",
        "db.m3.2xlarge"
      ]
    },
    "DatabaseAllocatedStorage": {
      "Description": "Database storage in GB; Default is set to 100GB, Minimum is 5GB
and Maximum is 6144GB",
      "Type": "Number",
      "MinValue": "5",
```

```
      "MaxValue": "6144",
      "Default": "100",
      "ConstraintDescription": "The storage allocated for the database must be between
5 and 6144 GB; at least 100 GB recommended"
    }
  },
  "Conditions": {
    "AlwaysTrue": {
      "Fn::Equals": [
        true,
        true
      ]
    },
    "AddIPRangeSSH": {
      "Fn::Not": [
        {
          "Fn::Equals": [
            {
              "Ref": "IPRangeSSH"
            },
            ""
          ]
        }
      ]
    },
    "AddIPRangeApplication": {
      "Fn::Not": [
        {
          "Fn::Equals": [
            {
              "Ref": "IPRangeApplication"
            },
            ""
          ]
        }
      ]
    }
  },
  "Mappings": {
    "RegionMap": {
      "us-east-1": {
        "AMI": "ami-8899e19e"
      },
      "us-east-2": {
        "AMI": "ami-0dbd9a68"
      },
      "us-west-1": {
        "AMI": "ami-f6c6e796"
      },
      "us-west-2": {
        "AMI": "ami-2f44224f"
      },
      "ca-central-1": {
```

```
        "AMI": "ami-a566dac1"
      },
      "eu-west-1": {
        "AMI": "ami-ee898288"
      },
      "eu-west-2": {
        "AMI": "ami-196d7a7d"
      },
      "eu-central-1": {
        "AMI": "ami-3866bf57"
      },
      "ap-southeast-1": {
        "AMI": "ami-1b951278"
      },
      "ap-southeast-2": {
        "AMI": "ami-b52431d6"
      },
      "ap-northeast-1": {
        "AMI": "ami-cc655cab"
      },
      "ap-northeast-2": {
        "AMI": "ami-2d588543"
      },
      "ap-south-1": {
        "AMI": "ami-bb95e8d4"
      },
      "sa-east-1": {
        "AMI": "ami-4d3d5321"
      }
    }
  },
  "Outputs": {
    "URL": {
      "Description": "URL to application",
      "Value": {
        "Fn::Join": [
          "",
          [
            "http://",
            {
              "Fn::GetAtt": [
                "Machine",
                "PublicDnsName"
              ]
            }
          ]
        ]
      }
    },
    "PublicIp": {
      "Description": "Public IP address for WordPress application",
      "Value": {
        "Fn::GetAtt": [
```

```
          "Machine",
          "PublicIp"
        ]
      }
    },
    "PublicDnsName": {
      "Description": "Public DNS name for your WordPress application",
      "Value": {
        "Fn::GetAtt": [
          "Machine",
          "PublicDnsName"
        ]
      }
    }
  },
  "Resources": {
    "Vpc": {
      "Type": "AWS::EC2::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/16",
        "Tags": [
          {
            "Value": {
              "Ref": "AWS::StackName"
            },
            "Key": "Application"
          },
          {
            "Value": "Public",
            "Key": "Network"
          }
        ],
        "EnableDnsHostnames": "true",
        "EnableDnsSupport": "true"
      }
    },
    "PublicSubnet": {
      "Type": "AWS::EC2::Subnet",
      "Properties": {
        "CidrBlock": "10.0.4.0/22",
        "AvailabilityZone": {
          "Ref": "AvailabilityZone"
        },
        "Tags": [
          {
            "Value": {
              "Ref": "AWS::StackName"
            },
            "Key": "Application"
          },
          {
            "Value": "Public",
            "Key": "Network"
```

```
        }
      ],
      "VpcId": {
        "Ref": "Vpc"
      }
    }
  },
  "InternetGateway": {
    "Type": "AWS::EC2::InternetGateway",
    "DependsOn": "Vpc",
    "Properties": {
      "Tags": [
        {
          "Value": {
            "Ref": "AWS::StackName"
          },
          "Key": "Application"
        },
        {
          "Value": "Public",
          "Key": "Network"
        }
      ]
    }
  },
  "GatewayToInternet": {
    "Type": "AWS::EC2::VPCGatewayAttachment",
    "DependsOn": "InternetGateway",
    "Properties": {
      "InternetGatewayId": {
        "Ref": "InternetGateway"
      },
      "VpcId": {
        "Ref": "Vpc"
      }
    }
  },
  "PublicRouteTable": {
    "Type": "AWS::EC2::RouteTable",
    "Properties": {
      "Tags": [
        {
          "Value": {
            "Ref": "AWS::StackName"
          },
          "Key": "Application"
        },
        {
          "Value": "Public",
          "Key": "Network"
        }
      ],
      "VpcId": {
```

```
          "Ref": "Vpc"
        }
      }
    },
    "PublicRoute": {
      "Type": "AWS::EC2::Route",
      "DependsOn": "GatewayToInternet",
      "Properties": {
        "RouteTableId": {
          "Ref": "PublicRouteTable"
        },
        "DestinationCidrBlock": "0.0.0.0/0",
        "GatewayId": {
          "Ref": "InternetGateway"
        }
      }
    },
    "PublicSubnetRouteTableAssociation": {
      "Type": "AWS::EC2::SubnetRouteTableAssociation",
      "Properties": {
        "SubnetId": {
          "Ref": "PublicSubnet"
        },
        "RouteTableId": {
          "Ref": "PublicRouteTable"
        }
      }
    },
    "PrivateSubnet": {
      "Type": "AWS::EC2::Subnet",
      "Properties": {
        "CidrBlock": "10.0.0.0/22",
        "AvailabilityZone": {
          "Ref": "AvailabilityZone"
        },
        "Tags": [
          {
            "Value": {
              "Ref": "AWS::StackName"
            },
            "Key": "Application"
          },
          {
            "Value": "Private",
            "Key": "Network"
          }
        ],
        "VpcId": {
          "Ref": "Vpc"
        }
      }
    },
    "PrivateRouteTable": {
```

```
      "Type": "AWS::EC2::RouteTable",
      "Properties": {
        "Tags": [
          {
            "Value": {
              "Ref": "AWS::StackName"
            },
            "Key": "Application"
          },
          {
            "Value": "Public",
            "Key": "Network"
          }
        ],
        "VpcId": {
          "Ref": "Vpc"
        }
      }
    },
    "PrivateSubnetRouteTableAssociation": {
      "Type": "AWS::EC2::SubnetRouteTableAssociation",
      "Properties": {
        "SubnetId": {
          "Ref": "PrivateSubnet"
        },
        "RouteTableId": {
          "Ref": "PrivateRouteTable"
        }
      }
    },
    "PrivateRoute": {
      "Type": "AWS::EC2::Route",
      "Properties": {
        "InstanceId": {
          "Ref": "Machine"
        },
        "RouteTableId": {
          "Ref": "PrivateRouteTable"
        },
        "DestinationCidrBlock": "0.0.0.0/0"
      }
    },
    "SharedSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Security Group for the internal communications",
        "SecurityGroupIngress": [],
        "VpcId": {
          "Ref": "Vpc"
        }
      }
    },
    "SharedSecurityGroupIngress0": {
```

```
      "Type": "AWS::EC2::SecurityGroupIngress",
      "DependsOn": "SharedSecurityGroup",
      "Properties": {
        "IpProtocol": "-1",
        "FromPort": "3306",
        "ToPort": "3306",
        "SourceSecurityGroupId": {
          "Fn::GetAtt": [
            "SharedSecurityGroup",
            "GroupId"
          ]
        },
        "GroupId": {
          "Fn::GetAtt": [
            "SharedSecurityGroup",
            "GroupId"
          ]
        }
      }
    },
    "SecondaryPrivateSubnet": {
      "Type": "AWS::EC2::Subnet",
      "Properties": {
        "CidrBlock": "10.0.8.0/22",
        "AvailabilityZone": {
          "Ref": "SecondAvailabilityZone"
        },
        "Tags": [
          {
            "Value": {
              "Ref": "AWS::StackName"
            },
            "Key": "Application"
          },
          {
            "Value": "Private",
            "Key": "Network"
          }
        ],
        "VpcId": {
          "Ref": "Vpc"
        }
      }
    },
    "SecondaryPrivateSubnetRouteTableAssociation": {
      "Type": "AWS::EC2::SubnetRouteTableAssociation",
      "Properties": {
        "SubnetId": {
          "Ref": "SecondaryPrivateSubnet"
        },
        "RouteTableId": {
          "Ref": "PrivateRouteTable"
        }
```

```
      }
    },
    "FrontendSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Frontend instance security group",
        "SecurityGroupIngress": [
          {
            "Fn::If": [
              "AddIPRangeSSH",
              {
                "IpProtocol": "tcp",
                "FromPort": 22,
                "ToPort": 22,
                "CidrIp": {
                  "Ref": "IPRangeSSH"
                }
              },
              "AWS::NoValue"
            ]
          },
          {
            "Fn::If": [
              "AddIPRangeApplication",
              {
                "IpProtocol": "tcp",
                "FromPort": 80,
                "ToPort": 80,
                "CidrIp": {
                  "Ref": "IPRangeApplication"
                }
              },
              "AWS::NoValue"
            ]
          },
          {
            "Fn::If": [
              "AddIPRangeApplication",
              {
                "IpProtocol": "tcp",
                "FromPort": 443,
                "ToPort": 443,
                "CidrIp": {
                  "Ref": "IPRangeApplication"
                }
              },
              "AWS::NoValue"
            ]
          }
        ],
        "VpcId": {
          "Ref": "Vpc"
        }
```

```
      }
    },
    "DatabaseSecurityGroup": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "Database instance security group",
        "SecurityGroupIngress": [],
        "VpcId": {
          "Ref": "Vpc"
        }
      }
    },
    "Machine": {
      "Type": "AWS::EC2::Instance",
      "Metadata": {
        "AWS::CloudFormation::Init" : {
          "config": {
            "files": {
              "/tmp/secret": {
                "content": {
                  "Fn::Join": [
                    "",
                    [
                      "\n# PROVISIONER_PEER_PASSWORD=", {"Ref": "DatabasePassword"},
                      "\n# PROVISIONER_APP_PASSWORD=", {"Ref": "ApplicationPassword"}
                    ]
                  ]
                },
                "mode": "000644",
                "owner": "root",
                "group": "root"
              }
            }
          }
        }
      },
      "Description": "Frontend machine instance",
      "CreationPolicy": {
        "ResourceSignal": {
          "Timeout": "PT15M"
        }
      },
      "DependsOn": [
        "DatabaseInstance",
        "GatewayToInternet"
      ],
      "Properties": {
        "KeyName": {
          "Ref": "SSHKeyName"
        },
        "ImageId": {
          "Fn::FindInMap": [
            "RegionMap",
```

```
            {
              "Ref": "AWS::Region"
            },
            "AMI"
          ]
        },
        "InstanceType": {
          "Ref": "InstancesType"
        },
        "BlockDeviceMappings": [
          {
            "DeviceName": "/dev/xvdc",
            "Ebs": {
              "VolumeType": {
                "Ref": "InstancesDataDiskType"
              },
              "VolumeSize": {
                "Ref": "InstancesDataDiskSize"
              }
            }
          }
        ],
        "NetworkInterfaces": [
          {
            "GroupSet": [
              {
                "Ref": "FrontendSecurityGroup"
              },
              {
                "Ref": "SharedSecurityGroup"
              }
            ],
            "AssociatePublicIpAddress": "true",
            "SubnetId": {
              "Ref": "PublicSubnet"
            },
            "DeviceIndex": "0",
            "DeleteOnTermination": "true"
          }
        ],
        "SourceDestCheck": "false",
        "UserData": {
          "Fn::Base64": {
            "Fn::Join": [
              "",
              [
                "#!/bin/bash\n",
                "sudo cfn-init -v",
                " --stack ", {"Ref": "AWS::StackName"},
                " --resource Machine",
                " --region ", {"Ref": "AWS::Region"}, "\n",
                "\n# USER_DATA_FILE=/tmp/secret",
                "\n# PROVISIONER_PEER_PASSWORD_INPUT=", {"Ref": "AWS::AccountId"}, "
```

```
", {"Ref": "AWS::StackId"}, " ", {"Ref": "AWS::Region"},
                "\n# PROVISIONER_SHARED_UNIQUE_ID_INPUT=", {"Ref": "AWS::AccountId"},
" ", {"Ref": "AWS::StackId"}, " ", {"Ref": "AWS::Region"},
                "\n# PROVISIONER_CFN_RESOURCE=Machine",
                "\n# PROVISIONER_CFN_STACK=", {"Ref": "AWS::StackName"},
                "\n# PROVISIONER_CFN_REGION=", {"Ref": "AWS::Region"},
                "\n# PROVISIONER_DATA_DISK=/dev/xvdc",
                "\n# PROVISIONER_TIER=frontend",
                "\n# PROVISIONER_PEER_ADDRESS=", {"Fn::GetAtt": ["DatabaseInstance",
"Endpoint.Address"]}
              ]
            ]
          }
        },
        "Tags": [
          {
            "Value": "wordpress-frontend",
            "Key": "role"
          }
        ]
      }
    },
    "DatabaseSubnetGroup": {
      "Type": "AWS::RDS::DBSubnetGroup",
      "Properties": {
        "DBSubnetGroupDescription": "Database instance subnet group",
        "SubnetIds": [
          {
            "Ref": "PrivateSubnet"
          },
          {
            "Ref": "SecondaryPrivateSubnet"
          }
        ]
      }
    },
    "DatabaseInstance": {
      "Type": "AWS::RDS::DBInstance",
      "Properties": {
        "Engine": "mariadb",
        "StorageType": "gp2",
        "DBInstanceIdentifier": {
          "Fn::Join": [
            "",
            [
              {
                "Ref": "AWS::StackName"
              },
              "-DatabaseMachine"
            ]
          ]
        },
        "AvailabilityZone": {
```

```
          "Ref": "AvailabilityZone"
        },
        "DBSubnetGroupName": {
          "Ref": "DatabaseSubnetGroup"
        },
        "VPCSecurityGroups": [
          {
            "Ref": "DatabaseSecurityGroup"
          },
          {
            "Ref": "SharedSecurityGroup"
          }
        ],
        "DBInstanceClass": {
          "Ref": "DatabaseInstanceClass"
        },
        "AllocatedStorage": {
          "Ref": "DatabaseAllocatedStorage"
        },
        "MasterUsername": "root",
        "MasterUserPassword": {
          "Ref": "DatabasePassword"
        },
        "Tags": [
          {
            "Value": "wordpress-database",
            "Key": "role"
          }
        ]
      }
    },
    "DatabaseInstanceReplica": {
      "Type": "AWS::RDS::DBInstance",
      "Properties": {
        "Engine": "mariadb",
        "StorageType": "gp2",
        "DBInstanceIdentifier": {
          "Fn::Join": [
            "",
            [
              {
                "Ref": "AWS::StackName"
              },
              "-DatabaseMachineReplica"
            ]
          ]
        },
        "SourceDBInstanceIdentifier": {
          "Ref": "DatabaseInstance"
        },
        "AvailabilityZone": {
          "Ref": "SecondAvailabilityZone"
        },
```

```
        "VPCSecurityGroups": [
          {
            "Ref": "DatabaseSecurityGroup"
          },
          {
            "Ref": "SharedSecurityGroup"
          }
        ],
        "DBInstanceClass": {
          "Ref": "DatabaseInstanceClass"
        },
        "AllocatedStorage": {
          "Ref": "DatabaseAllocatedStorage"
        },
        "Tags": [
          {
            "Value": "wordpress-database",
            "Key": "role"
          }
        ]
      }
    }
  }
}
```

ANEXO D: PLANTILLAS DE CLOUDFORMATION

- **D.1 templates/rdsaurora.template**
- **D.2 templates/securitygroups.template**
- **D.3 templates/webserver.template**
- **D.4 templates/wordpress-master.template**
- **D.5 templates/wordpress.template**
- **D.6 ci/taskcat.yml**
- **D.7 ci/wordpress-master.json**

D.1 templates/rdsaurora.template

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "WordPress RDS Aurora-MySQL Template. (qs-??????)",
    "Parameters": {
        "Subnets": {
            "ConstraintDescription": "Must be list of existing subnet Ids",
            "Description": "At least two existing Subnets in separate Availability
Zones your Virtual Private Cloud (VPC)",
            "Type": "List<AWS::EC2::Subnet::Id>"
        },
        "AuroraRDSSecurityGroup": {
            "Description": "Aurora Security Group",
            "Type": "AWS::EC2::SecurityGroup::Id"
        },
        "DBAutoMinorVersionUpgrade": {
            "AllowedValues": [
                "true",
                "false"
            ],
            "Default": "true",
            "Description": "Select true/false to setup Auto Minor Version upgrade",
            "Type": "String"
        },
        "DBBackupRetentionPeriod": {
            "ConstraintDescription": "Database backup retention period must be between
1 and 35 days",
            "Default": "7",
            "Description": "The number of days for which automatic DB snapshots are
retained",
            "MaxValue": "35",
            "MinValue": "1",
```

```
            "Type": "Number"
        },
        "DBPreferredBackupWindow": {
            "AllowedPattern": "^(|([0-1][0-9]|2[0-3]):[0-5][0-9]-([0-1][0-9]|2[0-
3]):[0-5][0-9])$",
            "ConstraintDescription": "Preferred backup window must be left blank or in
the form of HH:MM-HH:MM",
            "Default": "",
            "Description": "(Optional) Preferred backup window",
            "Type": "String"
        },
        "DBInstanceClass": {
            "AllowedValues": [
                "db.t2.micro",
                "db.t2.small",
                "db.t2.medium",
                "db.t2.large",
                "db.t2.xlarge",
                "db.t2.2xlarge",
                "db.m3.medium",
                "db.m3.large",
                "db.m3.xlarge",
                "db.m3.2xlarge",
                "db.m4.large",
                "db.m4.xlarge",
                "db.m4.2xlarge",
                "db.m4.4xlarge",
                "db.m4.10xlarge",
                "db.m4.16xlarge",
                "db.r3.large",
                "db.r3.xlarge",
                "db.r3.2xlarge",
                "db.r3.4xlarge",
                "db.r3.8xlarge",
                "db.r4.large",
                "db.r4.xlarge",
                "db.r4.2xlarge",
                "db.r4.4xlarge",
                "db.r4.8xlarge",
                "db.r4.16xlarge"
            ],
            "ConstraintDescription": "Must select a valid database instance type.",
            "Default": "db.t2.small",
            "Description": "The name of the compute and memory capacity class of the
DB instance",
            "Type": "String"
        },
        "DBName": {
            "AllowedPattern": "[a-zA-Z][a-zA-Z0-9]*",
            "Default": "QuickstartAuroraDB",
            "Description": "Name of Aurora DB for WordPress Stack",
            "MaxLength": "64",
            "MinLength": "5",
```

```
            "Type": "String"
        },
        "DBMasterUserPassword": {
            "AllowedPattern": "(?=\\S)[^@\/\"\\r\\n\\t\\f\\s]*",
            "ConstraintDescription": "Min 8 alphanumeric. Cannot contain white space,
@, /, \"",
            "Description": "The database admin account password (username is 'root')",
            "MaxLength": "41",
            "MinLength": "8",
            "NoEcho": "True",
            "Type": "String"
        },
        "DBMultiAZ": {
            "AllowedValues": [
                "true",
                "false"
            ],
            "Default": "true",
            "Description": "Specifies if the database instance is a multiple
Availability Zone deployment",
            "Type": "String"
        }
    },
    "Conditions": {
        "CreateReadReplica": {
            "Fn::Equals": [
                {
                    "Ref": "DBMultiAZ"
                },
                "true"
            ]
        }
    },
    "Resources": {
        "AuroraDBSubnetGroup": {
            "Type": "AWS::RDS::DBSubnetGroup",
            "Properties": {
                "DBSubnetGroupDescription": "Subnets available for the RDS Aurora DB
Instance",
                "SubnetIds": {
                    "Ref": "Subnets"
                }
            }
        },
        "AuroraDBCluster": {
            "Type": "AWS::RDS::DBCluster",
            "Properties": {
                "BackupRetentionPeriod": {
                    "Ref": "DBBackupRetentionPeriod"
                },
                "PreferredBackupWindow": {
                    "Ref": "DBPreferredBackupWindow"
                },
```

```
                "DBSubnetGroupName": {
                    "Ref": "AuroraDBSubnetGroup"
                },
                "Engine": "aurora",
                "EngineVersion": "5.6",
                "MasterUsername": "root",
                "MasterUserPassword": {
                    "Ref": "DBMasterUserPassword"
                },
                "VpcSecurityGroupIds": [
                    {
                        "Ref": "AuroraRDSSecurityGroup"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "WordPress-Aurora-DB-Cluster"
                    }
                ]
            }
        },
        "AuroraDBPrimaryInstance": {
            "Type": "AWS::RDS::DBInstance",
            "Properties": {
                "Engine": "aurora",
                "EngineVersion": "5.6",
                "DBClusterIdentifier": {
                    "Ref": "AuroraDBCluster"
                },
                "DBInstanceClass": {
                    "Ref": "DBInstanceClass"
                },
                "DBSubnetGroupName": {
                    "Ref": "AuroraDBSubnetGroup"
                },
                "AutoMinorVersionUpgrade": {
                    "Ref": "DBAutoMinorVersionUpgrade"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "WordPress-Aurora-PrimaryDB"
                    }
                ]
            }
        },
        "AuroraDBSecondaryInstance": {
            "Type": "AWS::RDS::DBInstance",
            "Condition": "CreateReadReplica",
            "Properties": {
                "Engine": "aurora",
                "EngineVersion": "5.6",
```

```
                "DBClusterIdentifier": {
                    "Ref": "AuroraDBCluster"
                },
                "DBInstanceClass": {
                    "Ref": "DBInstanceClass"
                },
                "DBSubnetGroupName": {
                    "Ref": "AuroraDBSubnetGroup"
                },
                "AutoMinorVersionUpgrade": {
                    "Ref": "DBAutoMinorVersionUpgrade"
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "WordPress-Aurora-SecondaryDB"
                    }
                ]
            }
        }
    },
    "Outputs": {
        "AuroraEndPoints": {
            "Description": "Aurora Cluster Endpoint to connect",
            "Value": {
                "Fn::Join": [
                    "",
                    [
                        {
                            "Fn::GetAtt": [
                                "AuroraDBCluster",
                                "Endpoint.Address"
                            ]
                        },
                        ":",
                        {
                            "Fn::GetAtt": [
                                "AuroraDBCluster",
                                "Endpoint.Port"
                            ]
                        },
                        "/",
                        {
                            "Ref": "DBName"
                        }
                    ]
                ]
            }
        },
        "DBName": {
            "Description": "Aurora DBName",
            "Value": {
                "Ref": "DBName"
```

```
            }
        },
        "AuroraEndPointAddress": {
            "Description": "Aurora Endpoint to connect",
            "Value": {
                "Fn::Join": [
                    "",
                    [
                        {
                            "Fn::GetAtt": [
                                "AuroraDBCluster",
                                "Endpoint.Address"
                            ]
                        }
                    ]
                ]
            }
        },
        "AuroraEndPointPort": {
            "Description": "Aurora Endpoint to connect",
            "Value": {
                "Fn::Join": [
                    "",
                    [
                        {
                            "Fn::GetAtt": [
                                "AuroraDBCluster",
                                "Endpoint.Port"
                            ]
                        }
                    ]
                ]
            }
        }
    }
}
```

D.2 templates/securitygroups.template

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "WordPress Security Groups template. (qs-???????)",
    "Parameters": {
        "VPC": {
            "Description": "VPC-ID of your existing Virtual Private Cloud (VPC) where
you want to depoy RDS",
            "Type": "AWS::EC2::VPC::Id"
        },
        "VPCCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/([0-9]|[1-2][0-9]|3[0-
2]))$",
            "ConstraintDescription": "Must be a valid IP range in x.x.x.x/x notation",
            "Description": "The CIDR block for VPC",
            "Type": "String"
        },
        "ALBAccessCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/([0-9]|[1-2][0-9]|3[0-
2]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/x",
            "Description": "Allowed CIDR block for external web access to the
Application Load Balancer",
            "Type": "String"
        },
        "BastionSecurityGroupID": {
            "Description": "ID of the Bastion Security Group (e.g., sg-1d2c3b4a)",
            "Type": "AWS::EC2::SecurityGroup::Id"
        }
    },
    "Resources": {
        "AuroraRDSSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "Allow access to Aurora Port (AWS Quick Start)",
                "VpcId": {
                    "Ref": "VPC"
                },
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "3306",
                        "ToPort": "3306",
                        "SourceSecurityGroupId": {
                            "Fn::GetAtt": [
                                "WebServerSecurityGroup",
                                "GroupId"
                            ]
                        }
```

```
                }
            ],
            "SecurityGroupEgress": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": "80",
                    "ToPort": "80",
                    "CidrIp": "0.0.0.0/0"
                },
                {
                    "IpProtocol": "tcp",
                    "FromPort": "443",
                    "ToPort": "443",
                    "CidrIp": "0.0.0.0/0"
                }
            ]
        }
    },
    "ALBSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "ELB Security Group (AWS Quick Start)",
            "VpcId": {
                "Ref": "VPC"
            },
            "SecurityGroupIngress": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": "80",
                    "ToPort": "80",
                    "CidrIp": { "Ref": "ALBAccessCIDR" }
                },
                {
                    "IpProtocol": "tcp",
                    "FromPort": "443",
                    "ToPort": "443",
                    "CidrIp": { "Ref": "ALBAccessCIDR" }
                }
            ],
            "SecurityGroupEgress": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": "80",
                    "ToPort": "80",
                    "CidrIp": "0.0.0.0/0"
                },
                {
                    "IpProtocol": "tcp",
                    "FromPort": "443",
                    "ToPort": "443",
                    "CidrIp": "0.0.0.0/0"
                }
            ]
```

```
            }
        },
        "WebServerSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "Properties": {
                "GroupDescription": "App Server Security Group (AWS Quick Start)",
                "VpcId": {
                    "Ref": "VPC"
                },
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "22",
                        "ToPort": "22",
                        "SourceSecurityGroupId": {
                            "Ref": "BastionSecurityGroupID"
                        }
                    },
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "443",
                        "ToPort": "443",
                        "SourceSecurityGroupId": {
                            "Fn::GetAtt": [
                                "ALBSecurityGroup",
                                "GroupId"
                            ]
                        }
                    },
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "80",
                        "ToPort": "80",
                        "SourceSecurityGroupId": {
                            "Fn::GetAtt": [
                                "ALBSecurityGroup",
                                "GroupId"
                            ]
                        }
                    }
                ],
                "SecurityGroupEgress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "80",
                        "ToPort": "80",
                        "CidrIp": "0.0.0.0/0"
                    },
                    {
                        "IpProtocol": "tcp",
                        "FromPort": "443",
                        "ToPort": "443",
                        "CidrIp": "0.0.0.0/0"
```

```
                }
            ]
        }
    },
    "EFSSecurityGroup": {
        "Type": "AWS::EC2::SecurityGroup",
        "Properties": {
            "GroupDescription": "EFS Security Group",
            "VpcId": {
                "Ref": "VPC"
            },
            "SecurityGroupIngress": [
                {
                    "IpProtocol": "tcp",
                    "FromPort": "2049",
                    "ToPort": "2049",
                    "CidrIp": {
                        "Ref": "VPCCIDR"
                    }
                }
            ]
        }
    },
    "SecurityGroupRuleAppDB": {
        "Type": "AWS::EC2::SecurityGroupEgress",
        "Properties": {
            "IpProtocol": "tcp",
            "FromPort": "3306",
            "ToPort": "3306",
            "DestinationSecurityGroupId": {
                "Fn::GetAtt": [
                    "AuroraRDSSecurityGroup",
                    "GroupId"
                ]
            },
            "GroupId": {
                "Fn::GetAtt": [
                    "WebServerSecurityGroup",
                    "GroupId"
                ]
            }
        }
    },
    "SecurityGroupRuleAppEFS": {
        "Type": "AWS::EC2::SecurityGroupEgress",
        "Properties": {
            "IpProtocol": "tcp",
            "FromPort": "2049",
            "ToPort": "2049",
            "DestinationSecurityGroupId": {
                "Fn::GetAtt": [
                    "EFSSecurityGroup",
                    "GroupId"
```

```
                    ]
                },
                "GroupId": {
                    "Fn::GetAtt": [
                        "WebServerSecurityGroup",
                        "GroupId"
                    ]
                }
            }
        }
    },
    "Outputs": {
        "AuroraRDSSecurityGroup": {
            "Description": "Aurora Security Group",
            "Value": {
                "Ref": "AuroraRDSSecurityGroup"
            }
        },
        "ALBSecurityGroup": {
            "Description": "ELB Security Group",
            "Value": {
                "Ref": "ALBSecurityGroup"
            }
        },
        "WebServerSecurityGroup": {
            "Description": "Web Server Security Group",
            "Value": {
                "Ref": "WebServerSecurityGroup"
            }
        },
        "EFSSecurityGroup": {
            "Description": "EFS Security Group",
            "Value": {
                "Ref": "EFSSecurityGroup"
            }
        }
    }
}
```

D.3 templates/webserver.template

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "WordPress Webserver Template. (qs-???????)",
    "Metadata": {
        "AWS::CloudFormation::Interface": {
            "ParameterGroups": [
                {
                    "Label": {
                        "default": "Network Configuration"
                    },
                    "Parameters": [
                        "WebServerSubnets",
                        "ALBSecurityGroup",
                        "WebServerSecurityGroup",
                        "PublicSubnet1ID",
                        "PublicSubnet2ID",
                        "VPCID",
                        "EFSSecurityGroup"
                    ]
                },
                {
                    "Label": {
                        "default": "Database Configuration"
                    },
                    "Parameters": [
                        "DBEndpointAddress",
                        "DBMasterUserPassword"
                    ]
                },
                {
                    "Label": {
                        "default": "DNS and SSL Configuration"
                    },
                    "Parameters": [
                        "DomainName",
                        "SSLCertificateId",
                        "HostedZoneID"
                    ]
                },
                {
                    "Label": {
                        "default": "WordPress Webserver Configuration"
                    },
                    "Parameters": [
                        "WordpressAdminPassword",
                        "WebServerInstanceType",
                        "WebServerInstanceMonitoring",
                        "AutoScalingNotificationEmail",
                        "WebServerMinSize",
                        "WebServerMaxSize",
                        "WebServerDesiredCapacity",
```

```
                        "KeyPairName"
                    ]
                },
                {
                    "Label": {
                        "default": "AWS Quick Start Configuration"
                    },
                    "Parameters": [
                        "QSS3BucketName",
                        "QSS3KeyPrefix"
                    ]
                }
            ]
        }
    },
    "Parameters": {
        "QSS3BucketName": {
            "AllowedPattern": "^[0-9a-zA-Z]+([0-9a-zA-Z-]*[0-9a-zA-Z])*$",
            "ConstraintDescription": "AWS Quick Start bucket name can include numbers,
lowercase letters, uppercase letters, and hyphens (-). It cannot start or end with a
hyphen (-).",
            "Default": "quickstart-reference",
            "Description": "S3 bucket name for the AWS Quick Start assets. AWS Quick
Start bucket name can include numbers, lowercase letters, uppercase letters, and
hyphens (-). It cannot start or end with a hyphen (-)",
            "Type": "String"
        },
        "QSS3KeyPrefix": {
            "AllowedPattern": "^[0-9a-zA-Z-/]*$",
            "ConstraintDescription": "AWS Quick Start key prefix can include numbers,
lowercase letters, uppercase letters, hyphens (-), and forward slash (/). It cannot
start or end with forward slash (/) because they are automatically appended.",
            "Default": "wordpress/latest/",
            "Description": "S3 key prefix for the AWS Quick Start assets.AWS Quick
Start key prefix can include numbers, lowercase letters, uppercase letters, hyphens (-
), and forward slash (/). It cannot start or end with forward slash (/) because they
are automatically appended",
            "Type": "String"
        },
        "AutoScalingNotificationEmail": {
            "AllowedPattern": "([a-zA-Z0-9_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-
9]{1,3}\\.[0-9]{1,3}\\.)|(([a-zA-Z0-9\\-]+\\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\\]?)",
            "ConstraintDescription": "Must be a valid email address.",
            "Description": "Email address to notify Auto Scaling operations",
            "Type": "String"
        },
        "WebServerSubnets": {
            "ConstraintDescription": "Must be list of existing subnet Ids",
            "Default": "",
            "Description": "A list of subnet identifiers of Amazon VPCs where the
WebServer Autoscaling would be launched",
            "Type": "List<AWS::EC2::Subnet::Id>"
        },
```

```
        "EFSSecurityGroup": {
            "Description": "EFS Security Group",
            "Type": "AWS::EC2::SecurityGroup::Id"
        },
        "WebServerInstanceType": {
            "AllowedValues": [
                "t2.micro",
                "t2.small",
                "t2.medium",
                "t2.large",
                "t2.xlarge",
                "t2.2xlarge",
                "m3.medium",
                "m3.large",
                "m3.xlarge",
                "m3.2xlarge",
                "m4.large",
                "m4.xlarge",
                "m4.2xlarge",
                "m4.4xlarge",
                "m4.10xlarge",
                "m4.16xlarge",
                "m5.large",
                "m5.xlarge",
                "m5.2xlarge",
                "m5.4xlarge",
                "m5.12xlarge",
                "m5.24xlarge",
                "m5d.large",
                "m5d.xlarge",
                "m5d.2xlarge",
                "m5d.4xlarge",
                "m5d.12xlarge",
                "m5d.24xlarge",
                "c3.large",
                "c3.xlarge",
                "c3.2xlarge",
                "c3.4xlarge",
                "c3.8xlarge",
                "c4.large",
                "c4.xlarge",
                "c4.2xlarge",
                "c4.4xlarge",
                "c4.8xlarge",
                "c5.large",
                "c5.xlarge",
                "c5.2xlarge",
                "c5.4xlarge",
                "c5.9xlarge",
                "c5.18xlarge",
                "c5d.large",
                "c5d.xlarge",
                "c5d.2xlarge",
```

```
                "c5d.4xlarge",
                "c5d.9xlarge",
                "c5d.18xlarge",
                "r3.large",
                "r3.xlarge",
                "r3.2xlarge",
                "r3.4xlarge",
                "r3.8xlarge",
                "r4.large",
                "r4.xlarge",
                "r4.2xlarge",
                "r4.4xlarge",
                "r4.8xlarge",
                "r4.16xlarge",
                "i2.xlarge",
                "i2.2xlarge",
                "i2.4xlarge",
                "i2.8xlarge",
                "i3.large",
                "i3.xlarge",
                "i3.2xlarge",
                "i3.4xlarge",
                "i3.8xlarge",
                "i3.16xlarge",
                "x1.16xlarge",
                "x1.32xlarge",
                "x1e.xlarge",
                "x1e.2xlarge",
                "x1e.4xlarge",
                "x1e.8xlarge",
                "x1e.16xlarge",
                "x1e.32xlarge"
            ],
            "ConstraintDescription": "Choose an instance type.",
            "Default": "t2.small",
            "Description": "Select WordPress instance size",
            "Type": "String"
        },
        "WebServerInstanceMonitoring": {
            "Description": "Set enhanced monitoring for WordPress instances",
            "Type": "String",
            "Default": "Enabled",
            "AllowedValues": [
                "Enabled",
                "Disabled"
            ]
        },
        "WebServerSecurityGroup": {
            "Description": "Web Server Security Group",
            "Type": "AWS::EC2::SecurityGroup::Id"
        },
        "KeyPairName": {
            "ConstraintDescription": "Must be the name of an existing EC2 KeyPair.",
```

```
            "Default": "id_rsa_aws",
            "Description": "Name of an existing EC2 KeyPair to enable SSH access to
the instances. Use 'bitnami' user for the ssh connection to the WordPress instances",
            "Type": "AWS::EC2::KeyPair::KeyName"
        },
        "DBEndpointAddress": {
            "Description": "Aurora DB Endpoint",
            "Type": "String"
        },
        "DBMasterUserPassword": {
            "AllowedPattern": "(?=\\S)[^@/\"\\r\\n\\t\\f\\s]*",
            "ConstraintDescription": "Min 8 chars. Cannot contain white space, @, /,
\"",
            "Description": "The database admin account password (username is 'root')",
            "MaxLength": "41",
            "MinLength": "8",
            "NoEcho": "True",
            "Type": "String"
        },
        "WordpressAdminPassword": {
            "Description": "The WordPress site admin account password (username is
'user')",
            "Type": "String",
            "MinLength": "8",
            "MaxLength": "41",
            "AllowedPattern": "[a-zA-Z0-9]*",
            "ConstraintDescription": "Must contain only alphanumeric characters and
must be between 8 and 41 characters long.",
            "NoEcho": "true"
        },
        "WebServerMinSize": {
            "Default": "1",
            "Description": "Minimum number of WordPress instances in Auto Scaling
group",
            "Type": "Number"
        },
        "WebServerMaxSize": {
            "Default": "12",
            "Description": "Maximum number of WordPress instances in Auto Scaling
group",
            "Type": "Number"
        },
        "WebServerDesiredCapacity": {
            "Default": "2",
            "Description": "Desired number of WordPress instances in Auto Scaling
group",
            "Type": "Number"
        },
        "VPCID": {
            "Description": "Select the VPC to deploy WordPress",
            "Type": "AWS::EC2::VPC::Id"
        },
        "PublicSubnet1ID": {
```

```
            "Description": "Public Subnet ID 1 located in Availability Zone 1",
            "Type": "AWS::EC2::Subnet::Id"
        },
        "PublicSubnet2ID": {
            "Description": "Public Subnet ID 2 located in Availability Zone 2",
            "Type": "AWS::EC2::Subnet::Id"
        },
        "ALBSecurityGroup": {
            "Description": "ALB Security Group",
            "Type": "AWS::EC2::SecurityGroup::Id"
        },
        "DomainName": {
            "Description": "(Optional) Domain name for the web site",
            "Type": "String",
            "Default": "",
            "AllowedPattern": "^((\\w+\\.)+\\w+){0,1}$",
            "ConstraintDescription": "Must be a valid domain name."
        },
        "SSLCertificateId": {
            "Description": "(Optional) The ARN of the SSL certificate to use for the
load balancer. If not specified, the certificate will be auto-generated",
            "Type": "String",
            "Default": ""
        },
        "HostedZoneID": {
            "Description": "(Optional) Route 53 Hosted Zone ID of the domain name. If
left blank route 53 will not be configured and DNS must be setup manually. If you
specify this, you must also specify a Domain name",
            "Type": "String",
            "Default": "",
            "MaxLength": "32"
        }
    },
    "Mappings": {
        "AWSAMIRegionMap": {
            "us-east-1": {
                "BITNAMIWP": "ami-031f6afed3d897c71"
            },
            "us-east-2": {
                "BITNAMIWP": "ami-03a02f83490a69cbf"
            },
            "us-west-1": {
                "BITNAMIWP": "ami-0148383d8d34439d0"
            },
            "us-west-2": {
                "BITNAMIWP": "ami-08d52c79543040061"
            },
            "eu-central-1": {
                "BITNAMIWP": "ami-0fbb32e84a2d85f19"
            },
            "eu-west-1": {
                "BITNAMIWP": "ami-0245bf95d9daeb0b9"
            },
```

```
            "ap-southeast-2": {
                "BITNAMIWP": "ami-0bfa4ed58077ebabf"
            }
        }
    },
    "Rules": {
        "DomainNameIsPresentIfHostedZoneIDRule": {
            "Assertions": [
                {
                    "Assert": {
                        "Fn::Or": [
                          { "Fn::Equals" : [ { "Ref" : "HostedZoneID" }, "" ] },
                          { "Fn::Not": [ { "Fn::Equals" : [ { "Ref" : "DomainName" },
"" ] } ] } ]
                    },
                    "AssertDescription": "Please specify a 'Domain Name' if you
specify 'Route 53 Hosted Zone ID'"
                }
            ]
        }
    },
    "Conditions": {
        "UseSSL": {
            "Fn::Or": [
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "DomainName" }, "" ] } ] },
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "SSLCertificateId" }, "" ] }
] }
            ]
        },
        "GenerateSSLCertificate": {
            "Fn::And": [
                { "Condition": "UseSSL" },
                { "Fn::Equals": [ { "Ref": "SSLCertificateId" }, "" ] }
            ]
        },
        "AddDNSRecord": {
            "Fn::And": [
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "DomainName" }, "" ] } ] },
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "HostedZoneID" }, "" ] } ] }
            ]
        },
        "EnableInstanceMonitoring": {
            "Fn::Equals": [ { "Ref": "WebServerInstanceMonitoring" }, "Enabled" ]
        }
    },
    "Resources": {
        "WordpressEFS": {
            "Type": "AWS::EFS::FileSystem",
            "Properties": {
                "PerformanceMode": "generalPurpose",
                "Encrypted": true,
                "FileSystemTags": [
                    {
```

```
                    "Key": "Name",
                    "Value": "WordPress EFS Shared Filesystem"
                }
            ]
        }
    },
    "WordpressEFSMountTarget1": {
        "Type": "AWS::EFS::MountTarget",
        "Properties": {
            "FileSystemId": {
                "Ref": "WordpressEFS"
            },
            "SubnetId": {
                "Fn::Select": [
                    "0",
                    {
                        "Ref": "WebServerSubnets"
                    }
                ]
            },
            "SecurityGroups": [
                {
                    "Ref": "EFSSecurityGroup"
                }
            ]
        }
    },
    "WordpressEFSMountTarget2": {
        "Type": "AWS::EFS::MountTarget",
        "Properties": {
            "FileSystemId": {
                "Ref": "WordpressEFS"
            },
            "SubnetId": {
                "Fn::Select": [
                    "1",
                    {
                        "Ref": "WebServerSubnets"
                    }
                ]
            },
            "SecurityGroups": [
                {
                    "Ref": "EFSSecurityGroup"
                }
            ]
        }
    },
    "Route53RecordSet": {
        "Condition": "AddDNSRecord",
        "Type": "AWS::Route53::RecordSet",
        "Properties": {
            "Type": "A",
```

```
                "Name": { "Ref": "DomainName" },
                "AliasTarget": {
                    "HostedZoneId": { "Fn::GetAtt": [ "ApplicationLoadBalancer",
"CanonicalHostedZoneID" ] },
                    "DNSName": { "Fn::GetAtt": [ "ApplicationLoadBalancer", "DNSName"
] }
                },
                "HostedZoneId": { "Ref": "HostedZoneID" }
            }
        },
        "ACMCertificate": {
            "Condition": "GenerateSSLCertificate",
            "Type": "AWS::CertificateManager::Certificate",
            "Properties": {
                "DomainName": { "Ref": "DomainName" },
                "DomainValidationOptions": [ {
                    "DomainName": { "Ref": "DomainName" },
                    "ValidationDomain": { "Ref": "DomainName" }
                } ]
            }
        },
        "ApplicationLoadBalancer": {
            "Type": "AWS::ElasticLoadBalancingV2::LoadBalancer",
            "Properties": {
                "Subnets": [
                    {
                        "Ref": "PublicSubnet1ID"
                    },
                    {
                        "Ref": "PublicSubnet2ID"
                    }
                ],
                "SecurityGroups": [
                    {
                        "Ref": "ALBSecurityGroup"
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "WordpressALB"
                    }
                ]
            }
        },
        "ALBTargetGroup": {
            "Type": "AWS::ElasticLoadBalancingV2::TargetGroup",
            "Properties": {
                "HealthCheckIntervalSeconds": 30,
                "HealthCheckTimeoutSeconds": 5,
                "HealthyThresholdCount": 2,
                "HealthCheckPort": 80,
                "HealthCheckProtocol": "HTTP",
```

```
                "Port": 80,
                "Protocol": "HTTP",
                "UnhealthyThresholdCount": 5,
                "VpcId": {
                    "Ref": "VPCID"
                },
                "TargetGroupAttributes": [
                    {
                        "Key": "stickiness.enabled",
                        "Value": "true"
                    },
                    {
                        "Key": "stickiness.type",
                        "Value": "lb_cookie"
                    },
                    {
                        "Key": "stickiness.lb_cookie.duration_seconds",
                        "Value": "30"
                    }
                ]
            }
        },
        "ALBListener": {
            "Type": "AWS::ElasticLoadBalancingV2::Listener",
            "Properties": {
                "DefaultActions": [
                    {
                        "Type": "forward",
                        "TargetGroupArn": {
                            "Ref": "ALBTargetGroup"
                        }
                    }
                ],
                "LoadBalancerArn": {
                    "Ref": "ApplicationLoadBalancer"
                },
                "Port": { "Fn::If": [ "UseSSL", 443, 80 ] },
                "Protocol": { "Fn::If": [ "UseSSL", "HTTPS", "HTTP" ] },
                "Certificates": [
                    { "Fn::If": [ "UseSSL",
                        {
                            "CertificateArn": {
                                "Fn::If": [ "GenerateSSLCertificate",
                                    { "Ref": "ACMCertificate" },
                                    { "Ref": "SSLCertificateId" }
                                ]
                            }
                        },
                        { "Ref": "AWS::NoValue" }
                    ] }
                ]
            }
        },
```

```
        "WebServerASG": {
            "Type": "AWS::AutoScaling::AutoScalingGroup",
            "Properties": {
                "LaunchConfigurationName": {
                    "Ref": "WebServerLC"
                },
                "MinSize": {
                    "Ref": "WebServerMinSize"
                },
                "MaxSize": {
                    "Ref": "WebServerMaxSize"
                },
                "DesiredCapacity": {
                    "Ref": "WebServerDesiredCapacity"
                },
                "TargetGroupARNs": [
                    {
                        "Ref": "ALBTargetGroup"
                    }
                ],
                "VPCZoneIdentifier": {
                    "Ref": "WebServerSubnets"
                },
                "NotificationConfiguration": {
                    "TopicARN": {
                        "Ref": "NotificationTopic"
                    },
                    "NotificationTypes": [
                        "autoscaling:EC2_INSTANCE_LAUNCH",
                        "autoscaling:EC2_INSTANCE_LAUNCH_ERROR",
                        "autoscaling:EC2_INSTANCE_TERMINATE",
                        "autoscaling:EC2_INSTANCE_TERMINATE_ERROR"
                    ]
                },
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": {
                            "Fn::Sub": "${AWS::StackName}-Web Server"
                        },
                        "PropagateAtLaunch": "true"
                    }
                ]
            },
            "CreationPolicy": {
                "ResourceSignal": {
                    "Count": {
                        "Ref": "WebServerDesiredCapacity"
                    },
                    "Timeout": "PT15M"
                }
            },
            "UpdatePolicy" : {
```

```
                "AutoScalingRollingUpdate" : {
                    "MinInstancesInService" : 1,
                    "PauseTime" : "PT15M",
                    "SuspendProcesses" : [
                        "HealthCheck",
                        "ReplaceUnhealthy",
                        "AZRebalance",
                        "AlarmNotification",
                        "ScheduledActions"
                    ],
                    "WaitOnResourceSignals" : true
                }
              }
        },
        "WebServerLC": {
            "Type": "AWS::AutoScaling::LaunchConfiguration",
            "DependsOn": [
                "WordpressEFSMountTarget1",
                "WordpressEFSMountTarget2",
                "ApplicationLoadBalancer"
            ],
            "Metadata": {
                "AWS::CloudFormation::Init" : {
                    "config": {
                        "files": {
                            "/opt/bitnami/var/cfn-user-data": {
                                "content": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "\n# PROVISIONER_PEER_PASSWORD=", { "Ref":
"DBMasterUserPassword" },
                                            "\n# PROVISIONER_APP_PASSWORD=", { "Ref":
"WordpressAdminPassword" }
                                        ]
                                    ]
                                },
                                "mode": "000400",
                                "owner": "root",
                                "group": "root"
                            }
                        }
                    }
                }
            },
            "Properties": {
                "ImageId": {
                    "Fn::FindInMap": [
                        "AWSAMIRegionMap",
                        {
                            "Ref": "AWS::Region"
                        },
                        "BITNAMIWP"
```

```
                    ]
                },
                "UserData": {
                    "Fn::Base64": {
                        "Fn::Join": [
                            "",
                            [
                                "#!/bin/bash\n",
                                "\n# PROVISIONER_CFN_INIT_ENABLED=true",
                                "\n# PROVISIONER_CFN_INIT_RESOURCE=WebServerLC",
                                "\n#
PROVISIONER_CFN_INIT_FILE_PATH=/opt/bitnami/var/cfn-user-data",
                                "\n# PROVISIONER_CFN_RESOURCE=WebServerASG",
                                "\n# PROVISIONER_CFN_STACK=", { "Ref":
"AWS::StackName" },
                                "\n# PROVISIONER_CFN_REGION=", { "Ref": "AWS::Region"
},
                                "\n# PROVISIONER_PEER_PASSWORD_INPUT=", { "Ref":
"AWS::AccountId" },
                                " ", { "Ref": "AWS::StackId" },
                                " ", { "Ref": "AWS::Region" },
                                "\n# PROVISIONER_SHARED_UNIQUE_ID_INPUT=", { "Ref":
"AWS::AccountId" },
                                " ", { "Ref": "AWS::StackId" },
                                " ", { "Ref": "AWS::Region" },
                                "\n# PROVISIONER_EFS_URI=", {
                                    "Fn::Sub":
"${WordpressEFS}.efs.${AWS::Region}.amazonaws.com:/"
                                },
                                "\n# PROVISIONER_PUBLIC_URL=", { "Fn::If": [ "UseSSL",
"https://", "http://" ] },
                                { "Fn::If": [ "AddDNSRecord",
                                    { "Ref": "DomainName" },
                                    { "Fn::GetAtt": [ "ApplicationLoadBalancer",
"DNSName" ] }
                                ] },
                                "\n# PROVISIONER_TIER=frontend_ha",
                                "\n# PROVISIONER_PEER_ADDRESS=", { "Ref":
"DBEndpointAddress" }
                            ]
                        ]
                    }
                },
                "InstanceType": {
                    "Ref": "WebServerInstanceType"
                },
                "InstanceMonitoring": {
                    "Fn::If": [ "EnableInstanceMonitoring", true, false ]
                },
                "SecurityGroups": [
                    {
                        "Ref": "WebServerSecurityGroup"
                    }
```

```
                ],
                "KeyName": {
                    "Ref": "KeyPairName"
                }
            }
        },
        "NotificationTopic": {
            "Type": "AWS::SNS::Topic",
            "Properties": {
                "Subscription": [
                    {
                        "Endpoint": {
                            "Ref": "AutoScalingNotificationEmail"
                        },
                        "Protocol": "email"
                    }
                ],
                "TopicName": {
                    "Fn::Sub": "${AWS::StackName}-WordpressSnsTopic"
                }
            }
        },
        "WebServerTargetTrackingScalingPolicy": {
            "Type": "AWS::AutoScaling::ScalingPolicy",
            "Properties": {
                "AutoScalingGroupName": {
                    "Ref": "WebServerASG"
                },
                "Cooldown": "60",
                "PolicyType": "TargetTrackingScaling",
                "TargetTrackingConfiguration": {
                    "PredefinedMetricSpecification": {
                        "PredefinedMetricType": "ASGAverageCPUUtilization"
                    },
                    "TargetValue": 75.0
                }
            }
        }
    },
    "Outputs": {
        "APPURL": {
            "Condition": "AddDNSRecord",
            "Description": "The URL to access the web application",
            "Value": {
                "Fn::Join": [ "", [
                    { "Fn::If": [ "UseSSL", "https://", "http://" ] },
                    { "Ref": "DomainName" }
                ] ]
            }
        },
        "ELBURL": {
            "Description": "The URL of the ELB. Point your domain to it by using a
CNAME/ALIAS DNS record",
```

```
            "Value": {
                "Fn::Join": [ "", [
                    { "Fn::If": [ "UseSSL", "https://", "http://" ] },
                    { "Fn::GetAtt": [ "ApplicationLoadBalancer", "DNSName" ] }
                ] ]
            }
        }
    }
}
```

D.4 templates/wordpress-master.template

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "This template creates a new VPC and deploys WordPress. **WARNING**
This template creates EC2 instances and related resources. You will be billed for the
AWS resources used if you create a stack from this template. (qs-????????)",
    "Metadata": {
        "AWS::CloudFormation::Interface": {
            "ParameterGroups": [
                {
                    "Label": {
                        "default": "Network Configuration"
                    },
                    "Parameters": [
                        "AvailabilityZones",
                        "VPCCIDR",
                        "PrivateSubnet1CIDR",
                        "PrivateSubnet2CIDR",
                        "PublicSubnet1CIDR",
                        "PublicSubnet2CIDR",
                        "ALBAccessCIDR"
                    ]
                },
                {
                    "Label": {
                        "default": "Linux Bastion Configuration"
                    },
                    "Parameters": [
                        "BastionInstanceType",
                        "BastionAMIOS",
                        "RemoteAccessCIDR",
                        "KeyPairName"
                    ]
                },
                {
                    "Label": {
                        "default": "Amazon RDS (Aurora) Configuration"
                    },
                    "Parameters": [
                        "DBAutoMinorVersionUpgrade",
                        "DBBackupRetentionPeriod",
                        "DBPreferredBackupWindow",
                        "DBInstanceClass",
                        "DBMasterUserPassword",
                        "DBMultiAZ"
                    ]
                },
                {
                    "Label": {
                        "default": "DNS and SSL Configuration"
                    },
                    "Parameters": [
```

```
                    "DomainName",
                    "SSLCertificateId",
                    "HostedZoneID"
                ]
            },
            {
                "Label": {
                    "default": "WordPress Webserver Configuration"
                },
                "Parameters": [
                    "WordpressAdminPassword",
                    "WebServerInstanceType",
                    "WebServerInstanceMonitoring",
                    "WebServerMinSize",
                    "WebServerMaxSize",
                    "WebServerDesiredCapacity",
                    "AutoScalingNotificationEmail"
                ]
            },
            {
                "Label": {
                    "default": "AWS Quick Start Configuration"
                },
                "Parameters": [
                    "QSS3BucketName",
                    "QSS3KeyPrefix"
                ]
            }
        ],
        "ParameterLabels": {
            "AutoScalingNotificationEmail": {
                "default": "Autoscaling Notification Email"
            },
            "AvailabilityZones": {
                "default": "Availability Zones"
            },
            "BastionAMIOS": {
                "default": "Bastion AMI OS"
            },
            "BastionInstanceType": {
                "default": "Bastion Instance Type"
            },
            "DBAutoMinorVersionUpgrade": {
                "default": "Enable Auto Minor Version Upgrade"
            },
            "DBBackupRetentionPeriod": {
                "default": "Backup Retention Period"
            },
            "DBPreferredBackupWindow": {
                "default": "Preferred Backup Window"
            },
            "DBInstanceClass": {
                "default": "Database Instance Size"
```

```
        },
        "DBMasterUserPassword": {
            "default": "Database Admin Password"
        },
        "DBMultiAZ": {
            "default": "Multi-AZ Database"
        },
        "WordpressAdminPassword": {
            "default": "Admin Password"
        },
        "KeyPairName": {
            "default": "SSH KeyPair Name"
        },
        "PrivateSubnet1CIDR": {
            "default": "Private Subnet 1 CIDR"
        },
        "PrivateSubnet2CIDR": {
            "default": "Private Subnet 2 CIDR"
        },
        "PublicSubnet1CIDR": {
            "default": "Public Subnet 1 CIDR"
        },
        "PublicSubnet2CIDR": {
            "default": "Public Subnet 2 CIDR"
        },
        "ALBAccessCIDR": {
            "default": "Allowed CIDR for ALB Access"
        },
        "QSS3BucketName": {
            "default": "Quick Start S3 Bucket Name"
        },
        "QSS3KeyPrefix": {
            "default": "Quick Start S3 Key Prefix"
        },
        "RemoteAccessCIDR": {
            "default": "Allowed Bastion External Access CIDR"
        },
        "DomainName": {
            "default": "Domain Name"
        },
        "SSLCertificateId": {
            "default": "SSL certificate ARN"
        },
        "HostedZoneID": {
            "default": "Route 53 Hosted Zone ID"
        },
        "VPCCIDR": {
            "default": "VPC CIDR"
        },
        "WebServerDesiredCapacity": {
            "default": "Desired Number of Instances"
        },
        "WebServerInstanceType": {
```

```
                    "default": "Instance Size"
                },
                "WebServerInstanceMonitoring": {
                    "default": "Instance enhanced monitoring"
                },
                "WebServerMaxSize": {
                    "default": "Max Number of Instances"
                },
                "WebServerMinSize": {
                    "default": "Min Number of Instances"
                }
            }
        }
    },
    "Parameters": {
        "AutoScalingNotificationEmail": {
            "AllowedPattern": "([a-zA-Z0-9_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-
9]{1,3}\\.[0-9]{1,3}\\.)|(([a-zA-Z0-9\\-]+\\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\\]?)",
            "ConstraintDescription": "Must be a valid email address.",
            "Description": "Email address to notify Auto Scaling operations",
            "Type": "String"
        },
        "AvailabilityZones": {
            "Description": "List of Availability Zones to use for the subnets in the
VPC. Only two Availability Zones are used for this deployment, and the logical order
of your selections is preserved",
            "Type": "List<AWS::EC2::AvailabilityZone::Name>"
        },
        "BastionAMIOS": {
            "AllowedValues": [
                "Amazon-Linux-HVM",
                "CentOS-7-HVM",
                "Ubuntu-Server-14.04-LTS-HVM",
                "Ubuntu-Server-16.04-LTS-HVM"
            ],
            "Default": "Amazon-Linux-HVM",
            "Description": "The Linux distribution for the AMI to be used for the
bastion instances",
            "Type": "String"
        },
        "BastionInstanceType": {
            "AllowedValues": [
                "t2.nano",
                "t2.micro",
                "t2.small",
                "t2.medium",
                "t2.large",
                "m3.large",
                "m3.xlarge",
                "m3.2xlarge",
                "m4.large",
                "m4.xlarge",
                "m4.2xlarge",
```

```
                "m4.4xlarge"
            ],
            "Default": "t2.micro",
            "Description": "Amazon EC2 instance type for the bastion instances",
            "Type": "String"
        },
        "DBAutoMinorVersionUpgrade": {
            "AllowedValues": [
                "true",
                "false"
            ],
            "Default": "true",
            "Description": "Select true/false to setup Auto Minor Version upgrade",
            "Type": "String"
        },
        "DBBackupRetentionPeriod": {
            "ConstraintDescription": "Database backup retention period must be between
1 and 35 days",
            "Default": "7",
            "Description": "The number of days for which automatic DB snapshots are
retained",
            "MaxValue": "35",
            "MinValue": "1",
            "Type": "Number"
        },
        "DBPreferredBackupWindow": {
            "AllowedPattern": "^(|([0-1][0-9]|2[0-3]):[0-5][0-9]-([0-1][0-9]|2[0-
3]):[0-5][0-9])$",
            "ConstraintDescription": "Preferred backup window must be left blank or in
the form of HH:MM-HH:MM",
            "Default": "",
            "Description": "(Optional) Preferred backup window",
            "Type": "String"
        },
        "DBInstanceClass": {
            "AllowedValues": [
                "db.t2.micro",
                "db.t2.small",
                "db.t2.medium",
                "db.t2.large",
                "db.t2.xlarge",
                "db.t2.2xlarge",
                "db.m3.medium",
                "db.m3.large",
                "db.m3.xlarge",
                "db.m3.2xlarge",
                "db.m4.large",
                "db.m4.xlarge",
                "db.m4.2xlarge",
                "db.m4.4xlarge",
                "db.m4.10xlarge",
                "db.m4.16xlarge",
                "db.r3.large",
```

```
                "db.r3.xlarge",
                "db.r3.2xlarge",
                "db.r3.4xlarge",
                "db.r3.8xlarge",
                "db.r4.large",
                "db.r4.xlarge",
                "db.r4.2xlarge",
                "db.r4.4xlarge",
                "db.r4.8xlarge",
                "db.r4.16xlarge"
            ],
            "ConstraintDescription": "Must select a valid database instance type.",
            "Default": "db.t2.small",
            "Description": "Select Instance size for the Database",
            "Type": "String"
        },
        "DBMasterUserPassword": {
            "AllowedPattern": "(?=\\S)[^@/\"\\r\\n\\t\\f\\s]*",
            "ConstraintDescription": "Min 8 alphanumeric. Cannot contain white space,
@, /, \"",
            "Description": "The database admin account password (username is 'root')",
            "MaxLength": "41",
            "MinLength": "8",
            "NoEcho": "True",
            "Type": "String"
        },
        "DBMultiAZ": {
            "AllowedValues": [
                "true",
                "false"
            ],
            "Default": "true",
            "Description": "Specifies if the database instance is a multiple
Availability Zone deployment",
            "Type": "String"
        },
        "WordpressAdminPassword": {
            "AllowedPattern": "[a-zA-Z0-9]*",
            "ConstraintDescription": "Must contain only alphanumeric characters and
must be between 8 and 41 characters long.",
            "Description": "The WordPress site admin account password (username is
'user')",
            "MaxLength": "41",
            "MinLength": "8",
            "NoEcho": "true",
            "Type": "String"
        },
        "KeyPairName": {
            "Description": "The name of an existing public/private key pair, which
allows you to securely connect to your instance after it launches. Use 'bitnami' user
for the ssh connection to the WordPress instances",
            "Type": "AWS::EC2::KeyPair::KeyName"
        },
```

```
        "PrivateSubnet1CIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/(1[6-9]|2[0-8]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/16-28",
            "Default": "10.0.0.0/19",
            "Description": "CIDR block for private subnet 1 located in Availability
Zone 1",
            "Type": "String"
        },
        "PrivateSubnet2CIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/(1[6-9]|2[0-8]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/16-28",
            "Default": "10.0.32.0/19",
            "Description": "CIDR block for private subnet 2 located in Availability
Zone 2",
            "Type": "String"
        },
        "PublicSubnet1CIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/(1[6-9]|2[0-8]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/16-28",
            "Default": "10.0.128.0/20",
            "Description": "CIDR block for the public (DMZ) subnet 1 located in
Availability Zone 1",
            "Type": "String"
        },
        "PublicSubnet2CIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/(1[6-9]|2[0-8]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/16-28",
            "Default": "10.0.144.0/20",
            "Description": "CIDR block for the public (DMZ) subnet 2 located in
Availability Zone 2",
            "Type": "String"
        },
        "ALBAccessCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/([0-9]|[1-2][0-9]|3[0-
2]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/x",
            "Default": "0.0.0.0/0",
            "Description": "Allowed CIDR block for external web access to the
Application Load Balancer",
            "Type": "String"
        },
        "QSS3BucketName": {
            "AllowedPattern": "^[0-9a-zA-Z]+([0-9a-zA-Z-]*[0-9a-zA-Z])*$",
```

```
            "ConstraintDescription": "Quick Start bucket name can include numbers,
lowercase letters, uppercase letters, and hyphens (-). It cannot start or end with a
hyphen (-).",
            "Default": "quickstart-reference",
            "Description": "S3 bucket name for the Quick Start assets. This string can
include numbers, lowercase letters, uppercase letters, and hyphens (-). It cannot
start or end with a hyphen (-)",
            "Type": "String"
        },
        "QSS3KeyPrefix": {
            "AllowedPattern": "^[0-9a-zA-Z-/]*$",
            "ConstraintDescription": "AWS Quick Start key prefix can include numbers,
lowercase letters, uppercase letters, hyphens (-), and forward slash (/). It cannot
start or end with forward slash (/) because they are automatically appended.",
            "Default": "wordpress/latest/",
            "Description": "S3 key prefix for the AWS Quick Start assets.AWS Quick
Start key prefix can include numbers, lowercase letters, uppercase letters, hyphens (-
), and forward slash (/). It cannot start or end with forward slash (/) because they
are automatically appended",
            "Type": "String"
        },
        "RemoteAccessCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/([0-9]|[1-2][0-9]|3[0-
2]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/x",
            "Default": "0.0.0.0/0",
            "Description": "Allowed CIDR block for external SSH access to the
bastions",
            "Type": "String"
        },
        "DomainName": {
            "Description": "(Optional) Domain name for the web site",
            "Type": "String",
            "Default": "",
            "AllowedPattern": "^((\\w+\\.)+\\w+){0,1}$",
            "ConstraintDescription": "Must be a valid domain name."
        },
        "SSLCertificateId": {
            "Description": "(Optional) The ARN of the SSL certificate to use for the
load balancer. If not specified, the certificate will be auto-generated",
            "Type": "String",
            "Default": ""
        },
        "HostedZoneID": {
            "Description": "(Optional) Route 53 Hosted Zone ID of the domain name. If
left blank route 53 will not be configured and DNS must be setup manually. If you
specify this, you must also specify a Domain name",
            "Type": "String",
            "Default": "",
            "MaxLength": "32"
        },
```

```
        "VPCCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/(1[6-9]|2[0-8]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/16-28",
            "Default": "10.0.0.0/16",
            "Description": "CIDR block for the VPC",
            "Type": "String"
        },
        "WebServerDesiredCapacity": {
            "Default": "2",
            "Description": "Desired number of WordPress instances in the Auto Scaling
group",
            "Type": "Number"
        },
        "WebServerInstanceType": {
            "AllowedValues": [
                "t2.micro",
                "t2.small",
                "t2.medium",
                "t2.large",
                "t2.xlarge",
                "t2.2xlarge",
                "m3.medium",
                "m3.large",
                "m3.xlarge",
                "m3.2xlarge",
                "m4.large",
                "m4.xlarge",
                "m4.2xlarge",
                "m4.4xlarge",
                "m4.10xlarge",
                "m4.16xlarge",
                "m5.large",
                "m5.xlarge",
                "m5.2xlarge",
                "m5.4xlarge",
                "m5.12xlarge",
                "m5.24xlarge",
                "m5d.large",
                "m5d.xlarge",
                "m5d.2xlarge",
                "m5d.4xlarge",
                "m5d.12xlarge",
                "m5d.24xlarge",
                "c3.large",
                "c3.xlarge",
                "c3.2xlarge",
                "c3.4xlarge",
                "c3.8xlarge",
                "c4.large",
                "c4.xlarge",
                "c4.2xlarge",
```

```
                "c4.4xlarge",
                "c4.8xlarge",
                "c5.large",
                "c5.xlarge",
                "c5.2xlarge",
                "c5.4xlarge",
                "c5.9xlarge",
                "c5.18xlarge",
                "c5d.large",
                "c5d.xlarge",
                "c5d.2xlarge",
                "c5d.4xlarge",
                "c5d.9xlarge",
                "c5d.18xlarge",
                "r3.large",
                "r3.xlarge",
                "r3.2xlarge",
                "r3.4xlarge",
                "r3.8xlarge",
                "r4.large",
                "r4.xlarge",
                "r4.2xlarge",
                "r4.4xlarge",
                "r4.8xlarge",
                "r4.16xlarge",
                "i2.xlarge",
                "i2.2xlarge",
                "i2.4xlarge",
                "i2.8xlarge",
                "i3.large",
                "i3.xlarge",
                "i3.2xlarge",
                "i3.4xlarge",
                "i3.8xlarge",
                "i3.16xlarge",
                "x1.16xlarge",
                "x1.32xlarge",
                "x1e.xlarge",
                "x1e.2xlarge",
                "x1e.4xlarge",
                "x1e.8xlarge",
                "x1e.16xlarge",
                "x1e.32xlarge"
            ],
            "ConstraintDescription": "Choose an instance type.",
            "Default": "t2.small",
            "Description": "Select WordPress instance size",
            "Type": "String"
        },
        "WebServerInstanceMonitoring": {
            "Description": "Set enhanced monitoring for WordPress instances",
            "Type": "String",
            "Default": "Enabled",
```

```
            "AllowedValues": [
                "Enabled",
                "Disabled"
            ]
        },
        "WebServerMaxSize": {
            "Default": "12",
            "Description": "Maximum number of WordPress instances in the Auto Scaling
group",
            "Type": "Number"
        },
        "WebServerMinSize": {
            "Default": "1",
            "Description": "Minimum number of WordPress instances in the Auto Scaling
group",
            "Type": "Number"
        }
    },
    "Rules": {
        "DomainNameIsPresentIfHostedZoneIDRule": {
            "Assertions": [
                {
                    "Assert": {
                        "Fn::Or": [
                          { "Fn::Equals" : [ { "Ref" : "HostedZoneID" }, "" ] },
                          { "Fn::Not": [ { "Fn::Equals" : [ { "Ref" : "DomainName" },
"" ] } ] } ] }
                    },
                    "AssertDescription": "Please specify a 'Domain Name' if you
specify 'Route 53 Hosted Zone ID'"
                }
            ]
        },
        "EFSSupportedRegionRule": {
            "Assertions": [
                {
                    "Assert": {
                        "Fn::Contains": [
                            [
                                "us-east-1",
                                "us-east-2",
                                "us-west-1",
                                "us-west-2",
                                "eu-central-1",
                                "eu-west-1",
                                "ap-southeast-2"
                            ],
                            {
                                "Ref": "AWS::Region"
                            }
                        ]
                    },
                    "AssertDescription": "This Quick Start uses Amazon EFS which is
```

```
only available in the US East (N. Virginia), US East (Ohio), US West (N. California),
US West (Oregon), EU (Frankfurt), EU (Ireland) and Asia Pacific (Sydney) regions.
Please launch the stack in one of these regions"
                }
            ]
        }
    },
    "Conditions": {
        "AddDNSRecord": {
            "Fn::And": [
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "DomainName" }, "" ] } ] },
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "HostedZoneID" }, "" ] } ] }
            ]
        }
    },
    "Resources": {
        "VPCStack": {
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": {
                    "Fn::Sub":
"https://${QSS3BucketName}.s3.amazonaws.com/${QSS3KeyPrefix}submodules/quickstart-aws-
vpc/templates/aws-vpc.template"
                },
                "Parameters": {
                    "AvailabilityZones": {
                        "Fn::Join": [
                            ",",
                            {
                                "Ref": "AvailabilityZones"
                            }
                        ]
                    },
                    "KeyPairName": {
                        "Ref": "KeyPairName"
                    },
                    "NumberOfAZs": "2",
                    "PrivateSubnet1ACIDR": {
                        "Ref": "PrivateSubnet1CIDR"
                    },
                    "PrivateSubnet2ACIDR": {
                        "Ref": "PrivateSubnet2CIDR"
                    },
                    "PublicSubnet1CIDR": {
                        "Ref": "PublicSubnet1CIDR"
                    },
                    "PublicSubnet2CIDR": {
                        "Ref": "PublicSubnet2CIDR"
                    },
                    "VPCCIDR": {
                        "Ref": "VPCCIDR"
                    }
                }
```

```
            }
        },
        "BastionStack": {
            "DependsOn": "VPCStack",
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": {
                    "Fn::Sub":
"https://${QSS3BucketName}.s3.amazonaws.com/${QSS3KeyPrefix}submodules/quickstart-
linux-bastion/templates/linux-bastion.template"
                },
                "Parameters": {
                    "BastionAMIOS": {
                        "Ref": "BastionAMIOS"
                    },
                    "BastionInstanceType": {
                        "Ref": "BastionInstanceType"
                    },
                    "KeyPairName": {
                        "Ref": "KeyPairName"
                    },
                    "PublicSubnet1ID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.PublicSubnet1ID"
                        ]
                    },
                    "PublicSubnet2ID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.PublicSubnet2ID"
                        ]
                    },
                    "QSS3BucketName": {
                        "Ref": "QSS3BucketName"
                    },
                    "QSS3KeyPrefix": {
                        "Fn::Sub": "${QSS3KeyPrefix}submodules/quickstart-linux-
bastion/"
                    },
                    "RemoteAccessCIDR": {
                        "Ref": "RemoteAccessCIDR"
                    },
                    "VPCID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.VPCID"
                        ]
                    }
                }
            }
        },
        "WordPressStack": {
```

```
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": {
                    "Fn::Sub":
"https://${QSS3BucketName}.s3.amazonaws.com/${QSS3KeyPrefix}templates/wordpress.templa
te"
                },
                "Parameters": {
                    "VPCID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.VPCID"
                        ]
                    },
                    "VPCCIDR": {
                        "Ref": "VPCCIDR"
                    },
                    "BastionSecurityGroupID": {
                        "Fn::GetAtt": [
                            "BastionStack",
                            "Outputs.BastionSecurityGroupID"
                        ]
                    },
                    "PrivateSubnet1ID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.PrivateSubnet1AID"
                        ]
                    },
                    "PrivateSubnet2ID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.PrivateSubnet2AID"
                        ]
                    },
                    "PublicSubnet1ID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.PublicSubnet1ID"
                        ]
                    },
                    "PublicSubnet2ID": {
                        "Fn::GetAtt": [
                            "VPCStack",
                            "Outputs.PublicSubnet2ID"
                        ]
                    },
                    "DBInstanceClass": {
                        "Ref": "DBInstanceClass"
                    },
                    "DBMasterUserPassword": {
                        "Ref": "DBMasterUserPassword"
                    },
```

```
                    "DBMultiAZ": {
                        "Ref": "DBMultiAZ"
                    },
                    "DBAutoMinorVersionUpgrade": {
                        "Ref": "DBAutoMinorVersionUpgrade"
                    },
                    "DBBackupRetentionPeriod": {
                        "Ref": "DBBackupRetentionPeriod"
                    },
                    "DBPreferredBackupWindow": {
                        "Ref": "DBPreferredBackupWindow"
                    },
                    "WebServerInstanceType": {
                        "Ref": "WebServerInstanceType"
                    },
                    "KeyPairName": {
                        "Ref": "KeyPairName"
                    },
                    "WebServerInstanceMonitoring": {
                        "Ref": "WebServerInstanceMonitoring"
                    },
                    "WebServerMinSize": {
                        "Ref": "WebServerMinSize"
                    },
                    "WebServerMaxSize": {
                        "Ref": "WebServerMaxSize"
                    },
                    "WebServerDesiredCapacity": {
                        "Ref": "WebServerDesiredCapacity"
                    },
                    "AutoScalingNotificationEmail": {
                        "Ref": "AutoScalingNotificationEmail"
                    },
                    "DomainName": {
                        "Ref": "DomainName"
                    },
                    "SSLCertificateId": {
                        "Ref": "SSLCertificateId"
                    },
                    "HostedZoneID": {
                        "Ref": "HostedZoneID"
                    },
                    "WordpressAdminPassword": {
                        "Ref": "WordpressAdminPassword"
                    },
                    "QSS3BucketName": {
                        "Ref": "QSS3BucketName"
                    },
                    "QSS3KeyPrefix": {
                        "Ref": "QSS3KeyPrefix"
                    },
                    "ALBAccessCIDR": {
                        "Ref": "ALBAccessCIDR"
```

```
                    }
                }
            }
        }
    },
    "Outputs": {
        "APPURL": {
            "Condition": "AddDNSRecord",
            "Description": "The URL to access the web application",
            "Value": { "Fn::GetAtt": [ "WordPressStack", "Outputs.APPURL" ] }
        },
        "ELBURL": {
            "Description": "The URL of the ELB. Point your domain to it by using a
CNAME/ALIAS DNS record",
            "Value": { "Fn::GetAtt": [ "WordPressStack", "Outputs.ELBURL" ] }
        }
    }
}
```

D.5 templates/wordpress.template

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "This template deploys WordPress into an existing VPC. **WARNING**
This template creates EC2 instances and related resources. You will be billed for the
AWS resources used if you create a stack from this template. (qs-????????)",
    "Metadata": {
        "AWS::CloudFormation::Interface": {
            "ParameterGroups": [
                {
                    "Label": {
                        "default": "Network Configuration"
                    },
                    "Parameters": [
                        "VPCID",
                        "VPCCIDR",
                        "PrivateSubnet1ID",
                        "PrivateSubnet2ID",
                        "PublicSubnet1ID",
                        "PublicSubnet2ID",
                        "ALBAccessCIDR",
                        "BastionSecurityGroupID"
                    ]
                },
                {
                    "Label": {
                        "default": "Aurora Database Configuration"
                    },
                    "Parameters": [
                        "DBAutoMinorVersionUpgrade",
                        "DBBackupRetentionPeriod",
                        "DBPreferredBackupWindow",
                        "DBInstanceClass",
                        "DBMasterUserPassword",
                        "DBMultiAZ"
                    ]
                },
                {
                    "Label": {
                        "default": "DNS and SSL Configuration"
                    },
                    "Parameters": [
                        "DomainName",
                        "SSLCertificateId",
                        "HostedZoneID"
                    ]
                },
                {
                    "Label": {
                        "default": "WordPress Webserver Configuration"
                    },
                    "Parameters": [
```

```
                    "WordpressAdminPassword",
                    "WebServerInstanceType",
                    "WebServerInstanceMonitoring",
                    "KeyPairName",
                    "WebServerMinSize",
                    "WebServerMaxSize",
                    "WebServerDesiredCapacity",
                    "AutoScalingNotificationEmail"
                ]
            },
            {
                "Label": {
                    "default": "AWS Quick Start Configuration"
                },
                "Parameters": [
                    "QSS3BucketName",
                    "QSS3KeyPrefix"
                ]
            }
        ],
        "ParameterLabels": {
            "AutoScalingNotificationEmail": {
                "default": "Autoscaling Notification Email"
            },
            "BastionSecurityGroupID": {
                "default": "Bastion Security Group ID"
            },
            "DBAutoMinorVersionUpgrade": {
                "default": "Enable Auto Minor Version Upgrade"
            },
            "DBBackupRetentionPeriod": {
                "default": "Backup Retention Period"
            },
            "DBPreferredBackupWindow": {
                "default": "Preferred Backup Window"
            },
            "DBInstanceClass": {
                "default": "Database Instance Size"
            },
            "DBMasterUserPassword": {
                "default": "Database Admin Password"
            },
            "DBMultiAZ": {
                "default": "Multi-AZ Database"
            },
            "WordpressAdminPassword": {
                "default": "Admin Password"
            },
            "KeyPairName": {
                "default": "SSH Keypair Name"
            },
            "PrivateSubnet1ID": {
                "default": "Private Subnet-1 ID"
```

```
                },
                "PrivateSubnet2ID": {
                    "default": "Private Subnet-2 ID"
                },
                "PublicSubnet1ID": {
                    "default": "Public Subnet-1 ID"
                },
                "PublicSubnet2ID": {
                    "default": "Public Subnet-2 ID"
                },
                "ALBAccessCIDR": {
                    "default": "Allowed CIDR for ALB Access"
                },
                "QSS3BucketName": {
                    "default": "Quick Start S3 Bucket Name"
                },
                "QSS3KeyPrefix": {
                    "default": "Quick Start S3 Key Prefix"
                },
                "DomainName": {
                    "default": "Domain Name"
                },
                "SSLCertificateId": {
                    "default": "SSL certificate ARN"
                },
                "HostedZoneID": {
                    "default": "Route 53 Hosted Zone ID"
                },
                "VPCCIDR": {
                    "default": "VPC CIDR"
                },
                "VPCID": {
                    "default": "VPC ID"
                },
                "WebServerDesiredCapacity": {
                    "default": "Desired Number of Instances"
                },
                "WebServerInstanceType": {
                    "default": "Instance Size"
                },
                "WebServerInstanceMonitoring": {
                    "default": "Instance enhanced monitoring"
                },
                "WebServerMaxSize": {
                    "default": "Max Number of Instances"
                },
                "WebServerMinSize": {
                    "default": "Min Number of Instances"
                }
            }
        }
    },
    "Parameters": {
```

```
        "AutoScalingNotificationEmail": {
            "AllowedPattern": "([a-zA-Z0-9_\\-\\.]+)@((\\[[0-9]{1,3}\\.[0-
9]{1,3}\\.[0-9]{1,3}\\.)|(([a-zA-Z0-9\\-]+\\.)+))([a-zA-Z]{2,4}|[0-9]{1,3})(\\]?)",
            "ConstraintDescription": "Must be a valid email address.",
            "Description": "Email address to notify Auto Scaling operations",
            "Type": "String"
        },
        "BastionSecurityGroupID": {
            "Description": "ID of the Bastion Security Group (e.g., sg-1d2c3b4a)",
            "Type": "AWS::EC2::SecurityGroup::Id"
        },
        "DBAutoMinorVersionUpgrade": {
            "AllowedValues": [
                "true",
                "false"
            ],
            "Default": "true",
            "Description": "Select true/false to setup Auto Minor Version upgrade",
            "Type": "String"
        },
        "DBBackupRetentionPeriod": {
            "ConstraintDescription": "Database backup retention period must be between
1 and 35 days",
            "Default": "7",
            "Description": "The number of days for which automatic DB snapshots are
retained",
            "MaxValue": "35",
            "MinValue": "1",
            "Type": "Number"
        },
        "DBPreferredBackupWindow": {
            "AllowedPattern": "^(|([0-1][0-9]|2[0-3]):[0-5][0-9]-([0-1][0-9]|2[0-
3]):[0-5][0-9])$",
            "ConstraintDescription": "Preferred backup window must be left blank or in
the form of HH:MM-HH:MM",
            "Default": "",
            "Description": "(Optional) Preferred backup window",
            "Type": "String"
        },
        "DBInstanceClass": {
            "AllowedValues": [
                "db.t2.micro",
                "db.t2.small",
                "db.t2.medium",
                "db.t2.large",
                "db.t2.xlarge",
                "db.t2.2xlarge",
                "db.m3.medium",
                "db.m3.large",
                "db.m3.xlarge",
                "db.m3.2xlarge",
                "db.m4.large",
                "db.m4.xlarge",
```

```
                "db.m4.2xlarge",
                "db.m4.4xlarge",
                "db.m4.10xlarge",
                "db.m4.16xlarge",
                "db.r3.large",
                "db.r3.xlarge",
                "db.r3.2xlarge",
                "db.r3.4xlarge",
                "db.r3.8xlarge",
                "db.r4.large",
                "db.r4.xlarge",
                "db.r4.2xlarge",
                "db.r4.4xlarge",
                "db.r4.8xlarge",
                "db.r4.16xlarge"
            ],
            "ConstraintDescription": "Must select a valid database instance type.",
            "Default": "db.t2.small",
            "Description": "Select Instance size for the Database",
            "Type": "String"
        },
        "DBMasterUserPassword": {
            "AllowedPattern": "(?=\\S)[^@/\"\\r\\n\\t\\f\\s]*",
            "ConstraintDescription": "Min 8 alphanumeric. Cannot contain white space,
@, /, \"",
            "Description": "The database admin account password (username is 'root')",
            "MaxLength": "41",
            "MinLength": "8",
            "NoEcho": "True",
            "Type": "String"
        },
        "DBMultiAZ": {
            "AllowedValues": [
                "true",
                "false"
            ],
            "Default": "true",
            "Description": "Specifies if the database instance is a multiple
Availability Zone deployment",
            "Type": "String"
        },
        "WordpressAdminPassword": {
            "AllowedPattern": "[a-zA-Z0-9]*",
            "ConstraintDescription": "Must contain only alphanumeric characters and
must be between 8 and 41 characters long.",
            "Description": "The WordPress site admin account password (username is
'user')",
            "MaxLength": "41",
            "MinLength": "8",
            "NoEcho": "true",
            "Type": "String"
        },
        "KeyPairName": {
```

```
            "ConstraintDescription": "Must be the name of an existing EC2 KeyPair.",
            "Default": "id_rsa_aws",
            "Description": "Name of an existing EC2 KeyPair to enable SSH access to
the instances. Use 'bitnami' user for the ssh connection in the WordPress instances",
            "Type": "AWS::EC2::KeyPair::KeyName"
        },
        "PrivateSubnet1ID": {
            "Description": "Private Subnet ID 1 located in Availability Zone 1",
            "Type": "AWS::EC2::Subnet::Id"
        },
        "PrivateSubnet2ID": {
            "Description": "Private Subnet ID 2  located in Availability Zone 2",
            "Type": "AWS::EC2::Subnet::Id"
        },
        "PublicSubnet1ID": {
            "Description": "Public Subnet ID 1 located in Availability Zone 1",
            "Type": "AWS::EC2::Subnet::Id"
        },
        "PublicSubnet2ID": {
            "Description": "Public Subnet ID 2 located in Availability Zone 2",
            "Type": "AWS::EC2::Subnet::Id"
        },
        "ALBAccessCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/([0-9]|[1-2][0-9]|3[0-
2]))$",
            "ConstraintDescription": "CIDR block parameter must be in the form
x.x.x.x/x",
            "Default": "0.0.0.0/0",
            "Description": "Allowed CIDR block for external web access to the
Application Load Balancer",
            "Type": "String"
        },
        "QSS3BucketName": {
            "AllowedPattern": "^[0-9a-zA-Z]+([0-9a-zA-Z-]*[0-9a-zA-Z])*$",
            "ConstraintDescription": "Quick Start bucket name can include numbers,
lowercase letters, uppercase letters, and hyphens (-). It cannot start or end with a
hyphen (-).",
            "Default": "quickstart-reference",
            "Description": "S3 bucket name for the Quick Start assets. This string can
include numbers, lowercase letters, uppercase letters, and hyphens (-). It cannot
start or end with a hyphen (-)",
            "Type": "String"
        },
        "QSS3KeyPrefix": {
            "AllowedPattern": "^[0-9a-zA-Z-/]*$",
            "ConstraintDescription": "AWS Quick Start key prefix can include numbers,
lowercase letters, uppercase letters, hyphens (-), and forward slash (/). It cannot
start or end with forward slash (/) because they are automatically appended.",
            "Default": "wordpress/latest/",
            "Description": "S3 key prefix for the AWS Quick Start assets.AWS Quick
Start key prefix can include numbers, lowercase letters, uppercase letters, hyphens (-
), and forward slash (/). It cannot start or end with forward slash (/) because they
```

```
are automatically appended",
            "Type": "String"
        },
        "DomainName": {
            "Description": "(Optional) Domain name for the web site",
            "Type": "String",
            "Default": "",
            "AllowedPattern": "^((\\w+\\.)+\\w+){0,1}$",
            "ConstraintDescription": "Must be a valid domain name."
        },
        "SSLCertificateId": {
            "Description": "(Optional) The ARN of the SSL certificate to use for the
load balancer. If not specified, the certificate will be auto-generated",
            "Type": "String",
            "Default": ""
        },
        "HostedZoneID": {
            "Description": "(Optional) Route 53 Hosted Zone ID of the domain name. If
left blank route 53 will not be configured and DNS must be setup manually. If you
specify this, you must also specify a Domain name",
            "Type": "String",
            "Default": "",
            "MaxLength": "32"
        },
        "VPCCIDR": {
            "AllowedPattern": "^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-
5])\\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])(\\/([0-9]|[1-2][0-9]|3[0-
2]))$",
            "ConstraintDescription": "Must be a valid IP range in x.x.x.x/x notation",
            "Description": "The CIDR IP range of VPC",
            "Type": "String"
        },
        "VPCID": {
            "Description": "Select the VPC to deploy WordPress",
            "Type": "AWS::EC2::VPC::Id"
        },
        "WebServerDesiredCapacity": {
            "Default": "2",
            "Description": "Desired number of WordPress instances in Auto Scaling
group",
            "Type": "Number"
        },
        "WebServerInstanceType": {
            "AllowedValues": [
                "t2.micro",
                "t2.small",
                "t2.medium",
                "t2.large",
                "t2.xlarge",
                "t2.2xlarge",
                "m3.medium",
                "m3.large",
                "m3.xlarge",
```

```
                "m3.2xlarge",
                "m4.large",
                "m4.xlarge",
                "m4.2xlarge",
                "m4.4xlarge",
                "m4.10xlarge",
                "m4.16xlarge",
                "m5.large",
                "m5.xlarge",
                "m5.2xlarge",
                "m5.4xlarge",
                "m5.12xlarge",
                "m5.24xlarge",
                "m5d.large",
                "m5d.xlarge",
                "m5d.2xlarge",
                "m5d.4xlarge",
                "m5d.12xlarge",
                "m5d.24xlarge",
                "c3.large",
                "c3.xlarge",
                "c3.2xlarge",
                "c3.4xlarge",
                "c3.8xlarge",
                "c4.large",
                "c4.xlarge",
                "c4.2xlarge",
                "c4.4xlarge",
                "c4.8xlarge",
                "c5.large",
                "c5.xlarge",
                "c5.2xlarge",
                "c5.4xlarge",
                "c5.9xlarge",
                "c5.18xlarge",
                "c5d.large",
                "c5d.xlarge",
                "c5d.2xlarge",
                "c5d.4xlarge",
                "c5d.9xlarge",
                "c5d.18xlarge",
                "r3.large",
                "r3.xlarge",
                "r3.2xlarge",
                "r3.4xlarge",
                "r3.8xlarge",
                "r4.large",
                "r4.xlarge",
                "r4.2xlarge",
                "r4.4xlarge",
                "r4.8xlarge",
                "r4.16xlarge",
                "i2.xlarge",
```

```
                "i2.2xlarge",
                "i2.4xlarge",
                "i2.8xlarge",
                "i3.large",
                "i3.xlarge",
                "i3.2xlarge",
                "i3.4xlarge",
                "i3.8xlarge",
                "i3.16xlarge",
                "x1.16xlarge",
                "x1.32xlarge",
                "x1e.xlarge",
                "x1e.2xlarge",
                "x1e.4xlarge",
                "x1e.8xlarge",
                "x1e.16xlarge",
                "x1e.32xlarge"
            ],
            "ConstraintDescription": "Choose an instance type.",
            "Default": "t2.small",
            "Description": "Select WordPress instance size",
            "Type": "String"
        },
        "WebServerInstanceMonitoring": {
            "Description": "Set enhanced monitoring for WordPress instances",
            "Type": "String",
            "Default": "Enabled",
            "AllowedValues": [
                "Enabled",
                "Disabled"
            ]
        },
        "WebServerMaxSize": {
            "Default": "12",
            "Description": "Maximum number of WordPress instances in the Auto Scaling
group",
            "Type": "Number"
        },
        "WebServerMinSize": {
            "Default": "1",
            "Description": "Minimum number of WordPress instances in the Auto Scaling
group",
            "Type": "Number"
        }
    },
    "Rules": {
        "DomainNameIsPresentIfHostedZoneIDRule": {
            "Assertions": [
                {
                    "Assert": {
                        "Fn::Or": [
                          { "Fn::Equals" : [ { "Ref" : "HostedZoneID" }, "" ] },
                          { "Fn::Not": [ { "Fn::Equals" : [ { "Ref" : "DomainName" },
```

```
"" ] } ] } ]
                    },
                    "AssertDescription": "Please specify a 'Domain Name' if you
specify 'Route 53 Hosted Zone ID'"
                }
            ]
        },
        "EFSSupportedRegionRule": {
            "Assertions": [
                {
                    "Assert": {
                        "Fn::Contains": [
                            [
                                "us-east-1",
                                "us-east-2",
                                "us-west-1",
                                "us-west-2",
                                "eu-central-1",
                                "eu-west-1",
                                "ap-southeast-2"
                            ],
                            {
                                "Ref": "AWS::Region"
                            }
                        ]
                    },
                    "AssertDescription": "This Quick Start uses Amazon EFS which is
only available in the US East (N. Virginia), US East (Ohio), US West (N. California),
US West (Oregon), EU (Frankfurt), EU (Ireland) and Asia Pacific (Sydney) regions.
Please launch the stack in one of these regions"
                }
            ]
        }
    },
    "Conditions": {
        "AddDNSRecord": {
            "Fn::And": [
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "DomainName" }, "" ] } ] },
                { "Fn::Not": [ { "Fn::Equals": [ { "Ref": "HostedZoneID" }, "" ] } ] }
            ]
        }
    },
    "Resources": {
        "SecurityGroupsStack": {
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": {
                    "Fn::Sub":
"https://${QSS3BucketName}.s3.amazonaws.com/${QSS3KeyPrefix}templates/securitygroups.t
emplate"
                },
                "Parameters": {
                    "VPC": {
```

```
                        "Ref": "VPCID"
                    },
                    "VPCCIDR": {
                        "Ref": "VPCCIDR"
                    },
                    "ALBAccessCIDR": {
                        "Ref": "ALBAccessCIDR"
                    },
                    "BastionSecurityGroupID": {
                        "Ref": "BastionSecurityGroupID"
                    }
                }
            }
        },
        "RDSAuroraStack": {
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": {
                    "Fn::Sub":
"https://${QSS3BucketName}.s3.amazonaws.com/${QSS3KeyPrefix}templates/rdsaurora.templa
te"
                },
                "Parameters": {
                    "Subnets": {
                        "Fn::Join": [
                            ",",
                            [
                                {
                                    "Ref": "PrivateSubnet1ID"
                                },
                                {
                                    "Ref": "PrivateSubnet2ID"
                                }
                            ]
                        ]
                    },
                    "AuroraRDSSecurityGroup": {
                        "Fn::GetAtt": [
                            "SecurityGroupsStack",
                            "Outputs.AuroraRDSSecurityGroup"
                        ]
                    },
                    "DBAutoMinorVersionUpgrade": {
                        "Ref": "DBAutoMinorVersionUpgrade"
                    },
                    "DBBackupRetentionPeriod": {
                        "Ref": "DBBackupRetentionPeriod"
                    },
                    "DBPreferredBackupWindow": {
                        "Ref": "DBPreferredBackupWindow"
                    },
                    "DBInstanceClass": {
                        "Ref": "DBInstanceClass"
```

```
                    },
                    "DBMasterUserPassword": {
                        "Ref": "DBMasterUserPassword"
                    },
                    "DBMultiAZ": {
                        "Ref": "DBMultiAZ"
                    }
                }
            }
        },
        "WebserverStack": {
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": {
                    "Fn::Sub":
"https://${QSS3BucketName}.s3.amazonaws.com/${QSS3KeyPrefix}templates/webserver.templa
te"
                },
                "Parameters": {
                    "QSS3BucketName": {
                        "Ref": "QSS3BucketName"
                    },
                    "QSS3KeyPrefix": {
                        "Ref": "QSS3KeyPrefix"
                    },
                    "AutoScalingNotificationEmail": {
                        "Ref": "AutoScalingNotificationEmail"
                    },
                    "WebServerSubnets": {
                        "Fn::Join": [
                            ",",
                            [
                                {
                                    "Ref": "PrivateSubnet1ID"
                                },
                                {
                                    "Ref": "PrivateSubnet2ID"
                                }
                            ]
                        ]
                    },
                    "EFSSecurityGroup": {
                        "Fn::GetAtt": [
                            "SecurityGroupsStack",
                            "Outputs.EFSSecurityGroup"
                        ]
                    },
                    "WebServerInstanceType": {
                        "Ref": "WebServerInstanceType"
                    },
                    "WebServerSecurityGroup": {
                        "Fn::GetAtt": [
                            "SecurityGroupsStack",
```

```
                "Outputs.WebServerSecurityGroup"
            ]
        },
        "KeyPairName": {
            "Ref": "KeyPairName"
        },
        "DBEndpointAddress": {
            "Fn::GetAtt": [
                "RDSAuroraStack",
                "Outputs.AuroraEndPointAddress"
            ]
        },
        "DBMasterUserPassword": {
            "Ref": "DBMasterUserPassword"
        },
        "WordpressAdminPassword": {
            "Ref": "WordpressAdminPassword"
        },
        "WebServerMinSize": {
            "Ref": "WebServerMinSize"
        },
        "WebServerMaxSize": {
            "Ref": "WebServerMaxSize"
        },
        "WebServerInstanceMonitoring": {
            "Ref": "WebServerInstanceMonitoring"
        },
        "WebServerDesiredCapacity": {
            "Ref": "WebServerDesiredCapacity"
        },
        "PublicSubnet1ID": {
            "Ref": "PublicSubnet1ID"
        },
        "PublicSubnet2ID": {
            "Ref": "PublicSubnet2ID"
        },
        "VPCID": {
            "Ref": "VPCID"
        },
        "ALBSecurityGroup": {
            "Fn::GetAtt": [
                "SecurityGroupsStack",
                "Outputs.ALBSecurityGroup"
            ]
        },
        "DomainName": {
            "Ref": "DomainName"
        },
        "SSLCertificateId": {
            "Ref": "SSLCertificateId"
        },
        "HostedZoneID": {
            "Ref": "HostedZoneID"
```

```
                    }
                }
            }
        }
    },
    "Outputs": {
        "APPURL": {
            "Condition": "AddDNSRecord",
            "Description": "The URL to access the web application",
            "Value": { "Fn::GetAtt": [ "WebserverStack", "Outputs.APPURL" ] }
        },
        "ELBURL": {
            "Description": "The URL of the ELB. Point your domain to it by using a
CNAME/ALIAS DNS record",
            "Value": { "Fn::GetAtt": [ "WebserverStack", "Outputs.ELBURL" ] }
        }
    }
}
```

D.6 ci/taskcat.yml

```
global:
  marketplace-ami: false
  owner: abjimenez@bitnami.com
  qsname: quickstart-bitnami-wordpress
  regions:
    - ap-southeast-2
    - eu-central-1
    - eu-west-1
    - us-east-1
    - us-east-2
    - us-west-1
    - us-west-2
  reporting: true
tests:
  bitnami-wordpress:
    parameter_input: wordpress-master.json
    template_file: wordpress-master.template
    regions:
    - ap-southeast-2
    - eu-central-1
    - eu-west-1
    - us-east-1
    - us-east-2
    - us-west-2
```

D.7 ci/wordpress-master.json

```
[
    {
        "ParameterKey": "QSS3BucketName",
        "ParameterValue": "quickstart-bitnami-wordpress"
    },
    {
        "ParameterKey": "QSS3KeyPrefix",
        "ParameterValue": "quickstart-bitnami-wordpress/"
    },
    {
        "ParameterKey": "AutoScalingNotificationEmail",
        "ParameterValue": "foo@example.com"
    },
    {
        "ParameterKey": "AvailabilityZones",
        "ParameterValue": "$[taskcat_genaz_2]"
    },
    {
        "ParameterKey": "BastionAMIOS",
        "ParameterValue": "Amazon-Linux-HVM"
    },
```

```
    {
        "ParameterKey": "BastionInstanceType",
        "ParameterValue": "t2.micro"
    },
    {
        "ParameterKey": "DBAutoMinorVersionUpgrade",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "DBBackupRetentionPeriod",
        "ParameterValue": "7"
    },
    {
        "ParameterKey": "DBInstanceClass",
        "ParameterValue": "db.t2.small"
    },
    {
        "ParameterKey": "DBMasterUserPassword",
        "ParameterValue": "$[taskcat_genpass_16A]"
    },
    {
        "ParameterKey": "DBMultiAZ",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "WordpressAdminPassword",
        "ParameterValue": "$[taskcat_genpass_16A]"
    },
    {
        "ParameterKey": "KeyPairName",
        "ParameterValue": "$[taskcat_getkeypair]"
    },
    {
        "ParameterKey": "RemoteAccessCIDR",
        "ParameterValue": "10.0.0.0/16"
    },
    {
        "ParameterKey": "DomainName",
        "ParameterValue": ""
    },
    {
        "ParameterKey": "SSLCertificateId",
        "ParameterValue": ""
    },
    {
        "ParameterKey": "HostedZoneID",
        "ParameterValue": ""
    },
    {
        "ParameterKey": "VPCCIDR",
        "ParameterValue": "10.0.0.0/16"
    },
    {
```

```
        "ParameterKey": "WebServerDesiredCapacity",
        "ParameterValue": "2"
    },
    {
        "ParameterKey": "WebServerInstanceType",
        "ParameterValue": "t2.small"
    },
    {
        "ParameterKey": "WebServerMaxSize",
        "ParameterValue": "3"
    },
    {
        "ParameterKey": "WebServerMinSize",
        "ParameterValue": "2"
    },
    {
        "ParameterKey": "WebServerInstanceMonitoring",
        "ParameterValue": "Enabled"
    },
    {
        "ParameterKey": "ALBAccessCIDR",
        "ParameterValue": "10.0.0.0/16"
    },
    {
        "ParameterKey": "DBPreferredBackupWindow",
        "ParameterValue": ""
    }
]
```

ANEXO E: CONFIGURACIÓN DE JENKINS

- **E.1 multi-tier.xml.tpl**
- **E.2 common.groovy**
- **E.3 multi-tier.groovy**

E.1 multi-tier.xml.tpl

```
{{#if dependsOnExternalRepository}}
        <hudson.model.StringParameterDefinition>
          <name>REPOSITORY_URL</name>
          <description>Repository url for applications using templates from an
external git server.</description>
          <defaultValue>github.com/bitnami/quickstart-bitnami-wordpress</defaultValue>
        </hudson.model.StringParameterDefinition>
        <hudson.model.StringParameterDefinition>
          <name>REPOSITORY_BRANCH</name>
          <description>Repository branch.</description>
          <defaultValue>master</defaultValue>
        </hudson.model.StringParameterDefinition>
        <hudson.model.BooleanParameterDefinition>
          <name>USE_TEMPLATE_AMI</name>
          <description>Use the AMI specified in the template and don't perform build.
If REUSE_BUILD is specified, the AMI from the referenced job will be
used.</description>
          <defaultValue>false</defaultValue>
        </hudson.model.BooleanParameterDefinition>
{{/if}}
```

E.2 common.groovy

```
def clonePrivateGitHubRepository(repo_url, branch = 'master', pull_request = '',
credentialsId = 'github-bot') {
    withCredentials([
        usernamePassword(credentialsId: credentialsId, usernameVariable:
'GITHUB_USER', passwordVariable: 'GITHUB_PASSWORD')
    ]) {
        sh "git clone -b ${branch}
\"https://${GITHUB_USER}:${GITHUB_PASSWORD}@${repo_url}\" ."
        if (pull_request != '') {
```

```
            sh "git fetch origin refs/pull/${pull_request}/head:pr-${pull_request}"
            sh "git checkout pr-${pull_request}"
        }
    }
}
```

E.3 multi-tier.groovy

```
common = load 'common.groovy'

def requiresExternalTemplate() {
    def ret = false
    if (common.variableExists('REPOSITORY_URL')) {
        ret = REPOSITORY_URL != ''
    }
    return ret
}

def build() {
    def performBuild = REUSE_BUILD.isEmpty()
    def templateDir = ''
    if (requiresExternalTemplate()) {
        performBuild = REUSE_BUILD.isEmpty() && !params.USE_TEMPLATE_AMI
    }

    stage('Build Initialization') {
        initializeBuild()
    }

    // Main build steps
    common.runAndCleanup {
        stage('Packaging') {
            if (performBuild) {
                try {
                    createProvisionerBundle(params.SKIP_LICENSES)
                } finally {
                    archive 'provisioner-build-logs/*,projects/provisioner-
build/output/nami-provisioner-licenses.*'
                }
            } else {
                if (requiresExternalTemplate() && REUSE_BUILD.isEmpty()) {
                    echo "Skipping pack. Use provisioner bundle from template image"
                } else {
                    echo "Reusing provisioner bundle from build ${REUSE_BUILD}"
                }
            }
        }

        stage('Prepare template') {
            if (requiresExternalTemplate()) {
```

```
                dir("template-workspace") {
                    common.clonePrivateGitHubRepository(REPOSITORY_URL,
REPOSITORY_BRANCH)
                    templateDir = "template-workspace/templates"
                }
            }
        }

        withMultiTierCredentials {
            try {
                stage('Cloud images building') {
                    if (performBuild) {
                        buildCloudImage()
                    } else {
                        if (requiresExternalTemplate() && REUSE_BUILD.isEmpty()) {
                            echo "Reusing cloud image from template:
${REPOSITORY_URL}"
                        } else {
                            echo "Reusing cloud image generated during build
${REUSE_BUILD}"
                        }
                    }
                }
                stage('Cloud images and templates testing') {
                    if (requiresExternalTemplate()) {
                        if (performBuild) {
                            testCloudImage()
                        } else {
                            echo "Skipping cloud image test"
                        }
                        testTemplate(templateDir)
                    } else {
                        testImageAndTemplate()
                    }
                }
            } finally {
                archive 'bradmin-logs/*'
            }
        }
    }
}
```

ANEXO F: CONFIGURACIÓN DE BRADMIN

- **F.1 aws/deployment.rb**
- **F.2 aws/external_template.rb**
- **F.3 aws/base.rb**

F.1 aws/deployment.rb

```
    # Returns a list of load balancers matching the name of the deployment
    def load_balancers
      all_deployments = cloud_client.deployments.map(&:id)
      nested_deployments = all_deployments.grep(/#{escape(name)}/)

      load_balancer_physical_ids = []
      nested_deployments.each do |ndid|
        nd = cloud_client.find_deployment ndid
        srs = cloud_client.request :describe_stack_resources, stack_name: nd.id
        lb = srs[:stack_resources].find do |sr|
          sr[:resource_type] == 'AWS::ElasticLoadBalancing::LoadBalancer' ||
            sr[:resource_type] == 'AWS::ElasticLoadBalancingV2::LoadBalancer'
        end
        load_balancer_physical_ids << lb[:physical_resource_id] unless lb.nil?
      end

      cloud_client.request(
        :describe_load_balancers,
        load_balancer_arns: load_balancer_physical_ids
      )[:load_balancers]
    end
```

F.2 aws/external_template.rb

```
module Bitnami
  module Aws
    module ExternalTemplate
      # Uploads a template for a limited time and returns the HTTPS url of the file
      # This uploaded template can be used in other templates to have nested resources
      def upload_template_for_test_deployment(tpl_file)
        tpl_name = File.basename(tpl_file)
        s3 = Bitnami::S3::Autobuild.new
        s3path = 's3://bitnami-autobuild-ondemand/temporary/external-templates/' \
```

```
                  "#{SecureRandom.hex(4)}-#{tpl_name}"
        s3.upload_file tpl_file, s3path, max_retries: 3
        s3path
      end

      # Sets a deletion policy for AWS resources.
      # E.g. useful to avoid creating snapshots of RDS clusters for testing
deployments
      def set_deletion_policy(tpl, resource, policy)
        log.debug "Setting #{resource} Deletion Policy to '#{policy}'"
        tpl['Resources'][resource]['DeletionPolicy'] = policy
        tpl
      end

      # Sets or modifies a property of an AWS resource.
      # append option can be used for properties that are arrays
      def set_template_resource_property(tpl, resource, property, value, **options)
        log.debug "Overwriting '#{property}' with '#{value}' in resource
'#{resource}'"
        log.debug "Current properties: #{tpl['Resources'][resource]['Properties']}"
        if options[:append]
          tpl['Resources'][resource]['Properties'][property] << value
        else
          tpl['Resources'][resource]['Properties'][property] = value
        end
        tpl
      end

      def populate_external_template_parameters
        parameters = {}
        if main_stack.topology == 'high_availability'
          parameters["#{main_stack.key.capitalize}AdminPassword".to_sym] =
            test_application_password
        end
        parameters[:KeyPairName] = Bitnami::BuildSettings.key_pair.test
        parameters[:DBMasterUserPassword] = test_application_password
        parameters[:AutoScalingNotificationEmail] = 'foo@example.com'
        parameters[:DBPreferredBackupWindow] = ''
        parameters[:AvailabilityZones] = 'us-east-1a,us-east-1c'
        parameters[:QSS3KeyPrefix] = "#{main_stack.key}/latest/"
        parameters
      end

      # Obtains the template files and prepare it for testing
      def prepare_template(**options)
        FileUtils.rm_rf template_deployment_directory if Dir.exist?
template_deployment_directory
        FileUtils.mkdir_p template_deployment_directory
        FileUtils.cp_r(Dir.glob("#{options[:template_dir]}/*"),
template_deployment_directory)

        return unless options[:mark_as_test]
```

```
        template_files = {}
        {
          rdsaurora:      'rdsaurora.template',
          securitygroups: 'securitygroups.template',
          webserver:      'webserver.template',
          main:           "#{main_stack.key}.template",
          master:         "#{main_stack.key}-master.template"
        }.each { |k, f| template_files[k] = "#{template_deployment_directory}/#{f}" }
        templates = {}
        template_files.keys.each do |t|
          templates[t] = JSON.parse(File.read(template_files[t]))
        end

        log.info('Set Deletion Policy to "Delete" for database cluster')
        templates[:rdsaurora] = set_deletion_policy(templates[:rdsaurora],
                                                    'AuroraDBCluster', 'Delete')

        log.info('Opening SMTPS port to be able to test it')
        templates[:securitygroups] = set_template_resource_property(
          templates[:securitygroups], 'WebServerSecurityGroup', 'SecurityGroupEgress',
          { IpProtocol: 'tcp', FromPort: '465', ToPort: '465', CidrIp: '0.0.0.0/0' },
          append: true
        )

        find_built_image
        if image_id.present?
          templates[:webserver] =
set_template_resource_property(templates[:webserver],
                                                                'WebServerLC',
                                                                'ImageId',
                                                                image_id)
        end

        if options[:mark_as_test]
          log.info('Marking the template as test')
          templates[:webserver] = mark_template_as_test(templates[:webserver],
'WebServerLC')
        end

        s3 = Bitnami::S3::Autobuild.new
        # Create a temporary s3 url for each template
        {
          securitygroups: 'SecurityGroupsStack',
          rdsaurora: 'RDSAuroraStack',
          webserver: 'WebserverStack'
        }.each do |tpl, resource|
          File.write(template_files[tpl], JSON.pretty_generate(templates[tpl]))
          templates[:main] = set_template_resource_property(
            templates[:main], resource, 'TemplateURL',
            s3.get_https_url(upload_template_for_test_deployment(template_files[tpl]))
          )
        end
        File.write(template_files[:main], JSON.pretty_generate(templates[:main]))
```

```
        templates[:master] = set_template_resource_property(
          templates[:master], "#{main_stack.name}Stack", 'TemplateURL',
          s3.get_https_url(upload_template_for_test_deployment(template_files[:main]))
        )
        File.write(template_files[:master], JSON.pretty_generate(templates[:master]))
      end

      def mark_template_as_test(tpl, resource)
        log.debug "Setting bitnami_testing_mode=true user data in #{resource}"

tpl['Resources'][resource]['Properties']['UserData']['Fn::Base64']['Fn::Join'][1] <<
          "\n#bitnami_testing_mode=true"
        tpl
      end
    end
  end
end
```

F.3 aws/base.rb

```
module Bitnami
  module Aws
    class Base < Bitnami::Image

      # https://docs.aws.amazon.com/general/latest/gr/rande.html#elasticfilesystem-
region
      EFS_SUPPORTED_REGIONS = %w[us-east-1 us-east-2 us-west-1 us-west-2 eu-central-1
eu-west-1
                                 ap-northeast-1 ap-northeast-2 ap-southeast-1
                                 ap-southeast-2].freeze

      def supported_regions
        # NOTE: Regions with EFS (Elastic File System) support
        return EFS_SUPPORTED_REGIONS if requires_efs?
      end

      # Returns true if the solution uses EFS
      def requires_efs?
        main_stack.is_a?(Bitnami::NamiStack) &&
          main_stack.key =~ /wordpress/ &&
          main_stack.topology == 'high_availability'
      end

      # Updates the list of AMIs present in the original template file with the public
AMIs
      def overwrite_ami_mapping(image_info, output_dir, file)
        json = JSON.parse(File.read("#{output_dir}/#{file}"))
        raise 'Unable to find AMI map' unless json['Mappings']['RegionMap'].present?

        map = {}
        image_info.each do |region, i|
          map[region] = { AMI: i.image_id }
        end

        json['Mappings']['RegionMap'] = map
        json_contents = JSON.pretty_generate(json)
        File.write("#{output_dir}/#{file}", json_contents)
        json_contents
      end

      def release_multitier_template(image_info, output_dir, _options = {})
        if File.exist? "#{output_dir}/#{main_stack.key}-master.template"
          # Use alternative method to release templates; there are multiple files
          log.info 'Releasing Multi-Tier external template'
          overwrite_ami_mapping(image_info, output_dir, 'webserver.template')
          return super
        end

        json_contents = overwrite_ami_mapping(image_info, output_dir, 'template.json')

        log.info "URLs to template for #{main_stack.key} #{main_stack.version}:"
```

```
        topology = main_stack.template_generator.topology
        url_prefix = "files/cloudformation/#{main_stack.key}"
        template_fullname = "#{main_stack.version}/#{main_stack.build_id}/" \
                            "#{topology}/template.json"
        downloads_path = "#{url_prefix}/#{template_fullname}"
        s3_url = "#{Bitnami::BaseStack::BITNAMI_DOWNLOADS_PREFIX}/#{downloads_path}"

        Bitnami::S3::Autobuild.new.write(s3_url, json_contents, acl: 'public-read')
        log.info
"https://#{Bitnami::BaseStack::BITNAMI_DOWNLOADS_HOST}/#{downloads_path}"
        log.info "https://s3.amazonaws.com/bitnami-downloads-cf/#{downloads_path}"
        log.info ''
        # Return the AWS uploaded template (just a json in this case)
        { s3_template_url: s3_url, uploaded: true }
      end
    end
  end
end
```

ANEXO G: TESTS FUNCIONALES

- **G.1 functional/addComment.js**
- **G.2 functional/checkExistingComment.js**
- **G.3 functional/SMTPConfigure.js**
- **G.4 functional/SMTPCheck.js**
- **G.5 functional/includes.js**

G.1 functional/addComment.js

```
'use strict';

casper.test.begin('Add comment', function(test) {

casper.start(`http://${casper.urlData.host}:${casper.urlData.port}${casper.urlData.pre
fix}`);
  casper.login(casper.defaultUser, test);
  casper.visitSite(casper.urlData, test);
  casper.goToPublishedPost(casper.testDefinitions.posts[0], test);
  casper.addComment(casper.testDefinitions.comments[0], test);
  casper.run(function() { test.done(); });
});
```

G.2 functional/checkExistingComment.js

```
'use strict';

casper.test.begin('Check existing comment', function(test) {

casper.start(`http://${casper.urlData.host}:${casper.urlData.port}${casper.urlData.pre
fix}`);
  casper.visitSite(casper.urlData, test);
  casper.goToPublishedPost(casper.testDefinitions.posts[0], test);
  casper.then(function() {
    test.assertTextExists(casper.testDefinitions.comments[0], `Comment
"${casper.testDefinitions.comments[0]}" exists`);
  });
  casper.run(function() { test.done(); });
});
```

G.3 functional/SMTPConfigure.js

```
'use strict';

casper.test.begin('SMTP Configure Test', function(test) {

casper.start(`http://${casper.urlData.host}:${casper.urlData.port}${casper.urlData.pre
fix}`);
  casper.login(casper.defaultUser, test);
  casper.goToPlugins(test);
  casper.SMTPConfigure(casper.testDefinitions.SMTPConf,
casper.testDefinitions.newUsers[0], test);
  casper.SMTPCheck(test);
  casper.logout(test);

  casper.run(function() { test.done(); });
});
```

G.4 functional/SMTPCheck.js

```
'use strict';

casper.test.begin('SMTP Operation Test', function(test) {

casper.start(`http://${casper.urlData.host}:${casper.urlData.port}${casper.urlData.pre
fix}`);
  casper.login(casper.defaultUser, test);
  casper.SMTPCheck(test);
  casper.logout(test);

  casper.run(function() { test.done(); });
});
```

G.5 functional/includes.js

```
'use strict';

/**
 * Add a comment in current blog post
 * Call from: Blog post (logged in)
 * @param {String} text - Comment text
 * @param {Object} test - Test handler
 */
casper.addComment = function addComment(text, test) {
  casper.waitUntilVisible('textarea#comment', function() {
    test.assertVisible('textarea#comment', 'Comment input exists');
```

```
    this.captureScreen('commentInput');
    casper.echo(`Adding comment content "${text}"`);
    this.sendKeys('textarea#comment', text);
  });
  casper.waitUntilVisible('input#submit', function() {
    test.assertVisible('input#submit', 'Submit button exists');
    this.captureScreen('submitButton');
    this.click('input#submit');
  });
  casper.waitUntilVisible(x(`//div[@class="comment-content"]/p[.="${text}"]`),
function() {
    test.assertVisible(x(`//div[@class="comment-content"]/p[.="${text}"]`), 'Comment
added successfully');
    this.captureScreen('commentAddedSuccessfully');
  });
};


/**
 * Configure SMTP using WP-mail plugin
 * @param  {Object} SMTPConf          SMTP configuration data
 * @param  {String} SMTPConf.host     SMTP host
 * @param  {String} SMTPConf.port     SMTP port
 * @param  {String} SMTPConf.user     SMTP user
 * @param  {String} SMTPConf.password SMTP password
 * @param  {Object} user              User data
 * @param  {String} user.username     User name
 * @param  {String} user.email        User e-mail
 * @param  {Object} test              Test handler
 */
casper.SMTPConfigure = function SMTPConfigure(SMTPConf, user, test) {
  casper.waitUntilVisible('span.activate a[href*="wp-mail-smtp"]', function success()
{
    test.assertVisible('span.activate a[href*="wp-mail-smtp"]', 'Activate WP-Mail-SMTP
button is visible');
    this.captureScreen('activate');
    this.click('span.activate a[href*="wp-mail-smtp"]');

    casper.waitUntilVisible('span.deactivate a[href*="wp-mail-smtp"]', function() {
      test.assertVisible('span.deactivate a[href*="wp-mail-smtp"]', 'WP-Mail-SMTP
activated successfully');
      this.captureScreen('activated');
    });
  }, function failure() {
    test.assertVisible('span.deactivate a[href*="wp-mail-smtp"]', 'WP-Mail-SMTP
activated previously');
  }, 10000 * casper.timeoutMultiplier);

  casper.waitUntilVisible('span a[href*="options"][href*="wp-mail-smtp"]', function()
{
    test.assertVisible('span a[href*="options"][href*="wp-mail-smtp"]', 'Settings link
is visible');
    this.captureScreen('settings');
```

```
    this.click('span a[href*="options"][href*="wp-mail-smtp"]');
  });

  casper.waitUntilVisible('#wp-mail-smtp-setting-from_email', function() {
    test.assertVisible('#wp-mail-smtp-setting-from_email', 'Form is visible');
    this.captureScreen('formEmpty');
    this.click('#wp-mail-smtp-setting-mailer-smtp');
    this.captureScreen('otherSMTP');
    this.fillSelectors('form[method="POST"]', {
      '#wp-mail-smtp-setting-from_email': user.email,
      '#wp-mail-smtp-setting-from_name': user.username,
      '#wp-mail-smtp-setting-smtp-host': SMTPConf.host,
      '#wp-mail-smtp-setting-smtp-port': SMTPConf.port,
      '#wp-mail-smtp-setting-smtp-user': SMTPConf.user,
      '#wp-mail-smtp-setting-smtp-pass': SMTPConf.password,
    }, false);

    test.assertVisible('#wp-mail-smtp-setting-smtp-enc-ssl', 'SSL selector is
visible');
    test.assertVisible(
      '#wp-mail-smtp-setting-row-smtp-autotls span.wp-mail-smtp-setting-toggle-
switch',
      'Auto TLS switch is visible'
    );
    test.assertVisible(
      '#wp-mail-smtp-setting-row-smtp-auth span.wp-mail-smtp-setting-toggle-switch',
      'SMTP Auth switch is visible'
    );
    this.click('#wp-mail-smtp-setting-smtp-enc-ssl');
    this.click('#wp-mail-smtp-setting-row-smtp-autotls span.wp-mail-smtp-setting-
toggle-switch');
    this.click('#wp-mail-smtp-setting-row-smtp-auth span.wp-mail-smtp-setting-toggle-
switch');
    this.captureScreen('formFilled');
    test.assertVisible('button[type="submit"]', 'Submit button is visible');
    this.click('button[type="submit"]');
  });

  casper.waitUntilVisible('div#message', function() {
    test.assertVisible('div#message', 'Settings saved');
    this.captureScreen('saved');
  });
};

/**
 * Check SMTP sending a test mail
 * @param  {Object} test  Test handler
 */
casper.SMTPCheck = function SMTPCheck(test) {

casper.thenOpen(`http://${casper.urlData.host}:${casper.urlData.port}${casper.urlData.
prefix}`
    + '/wp-admin/options-general.php?page=wp-mail-smtp&tab=test');
```

```
  casper.waitUntilVisible('#wp-mail-smtp-setting-test_email', function() {
    test.assertVisible('#wp-mail-smtp-setting-test_email', 'Recipient field exists');
    this.sendKeys('#wp-mail-smtp-setting-test_email', casper.testDefinitions.email,
{keepFocus: true});
  });

  casper.waitUntilVisible('button[type="submit"]', function() {
    test.assertVisible('button[type="submit"]', 'Submit button is visible');
    this.captureScreen('smtpBefore');
    this.click('button[type="submit"]');
  });

  casper.waitUntilVisible('#message.notice-success', function() {
    test.assertVisible('#message.notice-success', 'Success message exists');
    this.captureScreen('smtpResult');
    test.pass('Email was successfully sent');
  });
};
```

ANEXO H: TESTS DE VERIFICACIÓN

- **H.1 tests/functions/remote.js**
- **H.2 tests/verification/shared_disk.js**
- **H.3 tests/verification/load_balancer.js**

H.1 tests/functions/remote.js

```
'use strict';

const getLoginUser = require('./system-user').getLoginUser;

/**
 * Run program in a remote machine.
 * @function tests.functions.remote~runProgram
 * @param {string} ip - IP address of the remote machine
 * @param {string} prog - Program to execute
 * @param {string|string[]} args - Arguments. It can be either a string or an array
containing them
 * @param {Object} opts - Options that runProgram will receive
 * @param {Object} [opts.loginUser] - The user to log in as on the remote machine
 * @example
 * // Get status of file in a remote machine
 * remote.runProgram('11.22.33.44', 'stat', 'file.cnf', {retrieveStdStreams: true});
 */
function runProgram(ip, prog, args, opts) {
  opts = _.defaults(opts || {}, {loginUser: getLoginUser()});
  const loginUser = opts.loginUser;
  delete opts.loginUser;

  const sshArgs = ['-A', '-o', 'StrictHostKeyChecking=no', `${loginUser}@${ip}`];
  return $os.runProgram('ssh', sshArgs.concat(prog, args), opts);
}

/**
 * Run program in a remote machine as root (with sudo).
 * @function tests.functions.remote~runProgramWithSudo
 * @param {string} ip - IP address of the remote machine
 * @param {string} prog - Program to execute
 * @param {string|string[]} args - Arguments. It can be either a string or an array
containing them
 * @param {Object} opts - Options that runProgram will receive
 * @param {Object} [opts.loginUser] - The user to log in as on the remote machine
 * @example
```

```
 * // Get status of file in a remote machine
 * remote.runProgramWithSudo('11.22.33.44', 'stat', 'file.cnf', {retrieveStdStreams:
true});
 */
function runProgramWithSudo(ip, prog, args, opts) {
  args.unshift('SSH_AUTH_SOCK=$SSH_AUTH_SOCK', prog); // Keep SSH_AUTH_SOCK env-var
when using sudo
  return runProgram(ip, 'sudo', args, opts);
}

/**
 * Run scp.
 * @function tests.functions.remote~scp
 * @param {String|String[]} src
 * @param {String} dest
 * @example
 * scp('file', 'bitnami@10.0.0.6:~/file');
 * scp(['file1', 'file2'], 'bitnami@10.0.0.6:/tmp/');
 */
function scp(src, dst) {
  const sshArgs = ['-r', '-o', 'StrictHostKeyChecking=no'];
  $os.runProgram('scp', sshArgs.concat(src, dst));
}

/**
 * Copy tests and run them in secondary nodes.
 * @function tests.functions.remote~testSecondaryNode
 * @param {string} node - IP addresses of the remote machines
 * @param {string[]} excludedFiles - Test files to exclude in the secondary nodes
 * @param {Object} options - Options passed to the testSecondaryNode method
 * @param {Object} [options.loginUser] - The user to log in as on the remote machine
 * @param {Object} [options.name] - Name of the nami module to test
 * @param {Object} [options.filePatterns] - Array of patterns to match against the
list of test files for inclusion
 * @param {Object} [options.env] - Object containing extra environment variables
 * @example
 * // Copy all the test files but the current file and gonit.js
 * remote.testSecondaryNode('10.0.0.6', [__filename, 'gonit.js']);
 * // Copy all the test files. Use petete as user to log in
 * remote.testSecondaryNode('10.0.0.6', [], {loginUser: 'petete'});
 */
function testSecondaryNode(node, excludedFiles, options) {
  const defaultEnvVars = _.pick(process.env, Object.keys(process.env).filter(k =>
k.match(/^APP_/)));
  options = _.defaults(options || {}, {
    loginUser: getLoginUser(),
    name: $app.name,
    filePatterns: '*',
    env: {},
    clean: false,
  });
  options.env = Object.assign(defaultEnvVars, options.env);
  const testDir = $file.join('/tmp', `test_${options.name}`);
```

```
  // We need to create the test folder if it doesn't exist yet
  runProgram(node, 'mkdir', ['-p', testDir], {loginUser: options.loginUser});
  const testFiles = $file.glob(options.filePatterns, {exclude: excludedFiles,
absolutize: true, cwd: __dirname});

  scp(testFiles, `${options.loginUser}@${node}:${testDir}`);

  // Build a command with the env-vars provided and 'nami test'
  let args = [];
  _.each(Object.keys(options.env), (k) => {
    args.push(`${k}=${options.env[k]}`);
  });
  args = args.concat(['nami', 'test', '--test-dir', testDir, options.name]);
  const prog = args.shift();

  // Execute `nami test` in the secondary node
  const res = runProgramWithSudo(node, prog, args, {loginUser: options.loginUser,
retrieveStdStreams: true});

  // Clean the test directory after the tests are executed
  if (options.clean) {
    runProgramWithSudo(node, 'rm', ['-r', testDir], {loginUser: options.loginUser});
  }
  return res;
}

module.exports = {
  scp,
  runProgram,
  runProgramWithSudo,
  testSecondaryNode,
};
```

H.2 tests/verification/shared_disk.js

```
'use strict';

const remote = require('./remote');
const nf = require('./node-functions');

const nodes = nf.nodesInfo().all;

/*
- Get the mount from the /etc/fstab
- Shared disk
  - Write a file and it should be accessible in the rest of instances
  - Check the fstab
  - Check df -h so see if it's mounted
*/

describe('The shared disk', function() {
  const testFile = `test_${Math.random().toString(36)
    .substr(2, 6)}`;
  const fileContent = 'The content of the file that should be shared';
  const fstabFile = '/etc/fstab';
  const mount = {
    point: '/bitnami',
    type: 'nfs4',
  };

  before(function() {
    $file.write($file.join(mount.point, testFile), fileContent);
  });
  after(function() {
    $file.delete(testFile);
  });

  it(`should have an entry in ${fstabFile} for ${mount.point}`, function() {
    expect($file.read(fstabFile)).to.match(new
RegExp(`${mount.point}\\s${mount.type}`));
  });

  it(`should be mounted on ${mount.point}`, function() {
    expect($os.runProgram('mount', '-l')).to.contain(`on ${mount.point} type
${mount.type}`);
  });

  describe('Replication of files', function() {
    _.each(nodes, (node) => {
      it(`node ${node} should replicate the file`, function() {
        this.timeout(5000);
        this.retries(4);
        $util.sleep(1);
        const content = remote.runProgram(node, 'cat', $file.join(mount.point,
testFile));
        expect(content).to.contain(fileContent);
```

```
      });
    });
  });
});
```

H.3 tests/verification/load_balancer.js

```
'use strict';

const remote = require('./remote');
const nf = require('./node-functions');
const http = require('./common');

const nodes = nf.nodesInfo();
const publicIP = process.env.APP_LOADBALANCERIP; // The load balancer public IP
const publicPort = '80'; // Assume it's 80

describe('Check load balancing', function() {
  describe('The load balancer distributes requests', function() {
    this.timeout(30000);
    const activeNodes = [];
    const serverFile = $file.join($app.installation.root, 'apache/conf/httpd.conf');
    let serverFileContent = '';
    const headerName = 'instance-ip';

    before(function() {
      serverFileContent = $file.read(serverFile);
      $file.append(serverFile, `Header set ${headerName} "%{SERVER_ADDR}e"`,
{atNewLine: true});
      _.each(nodes.secondaries, (node) => {
        remote.scp(serverFile, `bitnami@${node}:${serverFile}`);
      });
      _.each(nodes.all, (node) => {
        remote.runProgramWithSudo(node, 'service', ['bitnami', 'restart']);
      });
      $util.sleep(1); // Wait for node restarts
    });
    beforeEach(function(done) {
      // make a request to the public IP
      http.httpGet(`http://${publicIP}:${publicPort}`, function(res) {
        activeNodes.push(res.headers[headerName]);
        done();
      });
    });
    after(function() {
      $file.write(serverFile, serverFileContent);
      _.each(nodes.secondaries, (node) => {
        remote.scp(serverFile, `bitnami@${node}:${serverFile}`);
      });
      _.each(nodes.all, (node) => {
        remote.runProgramWithSudo(node, 'service', ['bitnami', 'restart']);
      });
      $util.sleep(1); // Wait for node restarts
    });
    _.each(nodes.all, (node) => {
      it(`node ${node} should send a response`, function() {
        this.retries(nodes.all.length * 2); // Make the test more reliable applying x2
multiplier
```

```
        expect(activeNodes).to.include(node);
      });
    });
  });
});
```